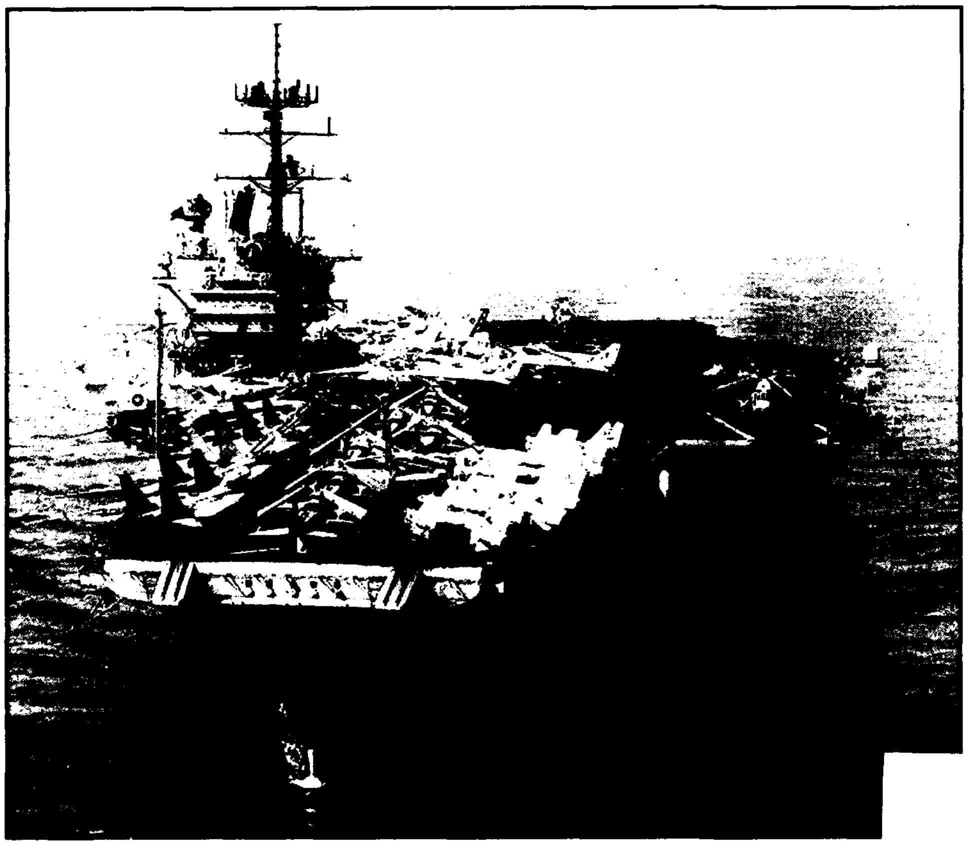
Technical Document 2332
March 1992

JDL Tri-Service Distributed Technology Experiment

L. R. Dunham
M. J. Gadbois
M. F. Barrett



Approved for public release; distribution is unlimited.

9

Technical Document 2332
March 1992

JDL Tri-Service Distributed Technology Experiment

L. R. Dunham
M. J. Gadbois
M. F. Barrett

NAVAL COMMAND, CONTROL AND OCEAN SURVEILLANCE CENTER RDT&E DIVISION
San Diego, California 92152-5000

J. D. FONTANA, CAPT, USN	R. T. SHEARER
Commanding Officer	**Executive Director**

ADMINISTRATIVE INFORMATION

This work was performed for the Office of Naval Technology under accession number DN888630, program element 62232N project CDB3.

Released by	Under authority of
M. C. Butterbrodt, Head	A. G. Justice, Head
Distributed Systems Branch	Information Processing
	and Displaying Division

LH

CONTENTS

1.0 INTRODUCTION

The Tri-Service Distributed Technology Experiment is an unclassified activity initiated under the auspices of the Joint Directors of Laboratories (JDL) by the Networks and Distributed Processing (N&DP) subpanel. The concept of the experiment centers around a demonstrable, short-term, and low-cost computer system application at three sites:

Naval Command, Control and Ocean Surveillance Center (NCCOSC), RDT&E Division (NRaD) San Diego, CA[1]
Rome Laboratory (RL), Rome, NY[2]
Communications-Electronics Command (CECOM), Ft. Monmouth, NJ.

The intention of the experiment is to overlap technology programs at each service and to use existing resources and internal manpower. The general objectives of the experiment are to emphasize activity among the three services (Navy, Air Force, and Army):

Service Cooperation:	Demonstrate that services can operate together to provide research support and development.
Service Data Sharing:	Demonstrate that service applications and data can be merged and shared in a distributed environment.
Service Resource Sharing:	Demonstrate that existing service resources can be tied together and shared for the benefit of all without superseding local control.
Interservice System Survivability:	Demonstrate that a distributed system resulting from merged resources can operate in a volatile environment.

The system used to support the Tri-Service Distributed Technology Experiment is the Cronus distributed computing environment. Cronus, developed by BBN Systems and Technologies Corporation, is a software system that provides an environment for the development and operation of distributed computing applications. It is designed specifically for a command and control (C2) environment. Cronus operates on a variety of different hardware bases and operating systems. It runs at user level, above the host's native operating system.

A JDL Cronus testbed was established at each site, using existing Cronus testbeds. The Tri-Service Distributed Technology Experiment was an integration of two Cronus applications running at NRaD and RL, along with new JDL components.

This document describes the functionalities of the tri-service components that make up the experiment and how to demonstrate the experiment. It is assumed that the user is familiar with the Cronus distributed computing environment and its debugging tool, *tropic*.

[1] NRaD was previously the Naval Ocean Systems Center (NOSC).
[2] Rome Laboratory was previously Rome Air Development Center (RADC).

Section 2.0 describes the components of experiment. Section 3.0 describes the setup and startup of the experiment demonstration.

Appendix A (A.1 through A.9) contains data object representation files for the experiment components.

Appendix B (B.1 through B.4) contains sample data files.

Appendix C (C.1 through C.6) contains sample command files for automatic startup.

2.0 EXPERIMENT COMPONENTS

The Tri-Service Distributed Technology Experiment consists of new JDL components and two existing applications (see figure 1): the Distributed Operating System Experiment (DOSE) designed and implemented by NRaD and the C2 Internet Experiment (C2Inet) developed by BBN and adapted by RL. The DOSE application provided three components to the experiment: Track Simulation Manager (Parser Manager), Track Report Manager, and Graphics Map Client. The Track Simulation Manager and Track Report Manager simulate Navy on-board ship scenarios, where radar track information is being logged and exchanged within a battle group. The Graphics Map Client displays the information graphically on a map. The C2Inet application provided four components to the experiment: Sensor Manager, Target Simulation Manager, Timer Manager, and Weather Manager. The Sensor Manager, Target Simulation Manager, and Weather Manager emulate an Air Force airborne or satellite observation post, where raw observations are generated, processed, and reported. The Timer Manager provides clock synchronization for the components. The components of these applications were adapted and integrated into the experiment.

CECOM contributed a new component, Situation Report Manager, which provided Army data for the experiment. RL implemented an additional new Air Force component, Target Filter Manager, to manage sensor data. Two new tri-service components, Simulation Data Manager and JDL Status Display Manager, were implemented to demonstrate that individual service data and resources can be shared in a distributed environment.

2.1 TRACK SIMULATION MANAGER

The Track Simulation Manager is an adaptation of the DOSE application component, Parser Manager. In the DOSE application, the primary function of the Parser Manager is to convert Link 11 binary messages into ASCII text. Due to the sensitive nature of the Link 11 data formats, there is a classified and an unclassified version of the Parser Manager. In the unclassified version, the operation of parsing blocks of binary data is not implemented. Instead, previously parsed track information is stored in a data file, *trackfile*, to be read by the Parser Manager. The track information is passed to the Track Report Manager for storage in its object database. The Parser Manager does not save information about the individual tracks. The Parser Manager is written in the C language and can run on UNIX and VMS (MicroVAX) machines.

The unclassified version of the Parser Manager was adapted to provide a demonstration scenerio of Navy data for the JDL Tri-Service Distributed Technology Experiment.

2.1.1 Integration of Track Simulation Manager

As described in section 2.1, the Track Simulation Manager is the unclassified version of Parser Manager adapted for the tri-service experiment. In the tri-service

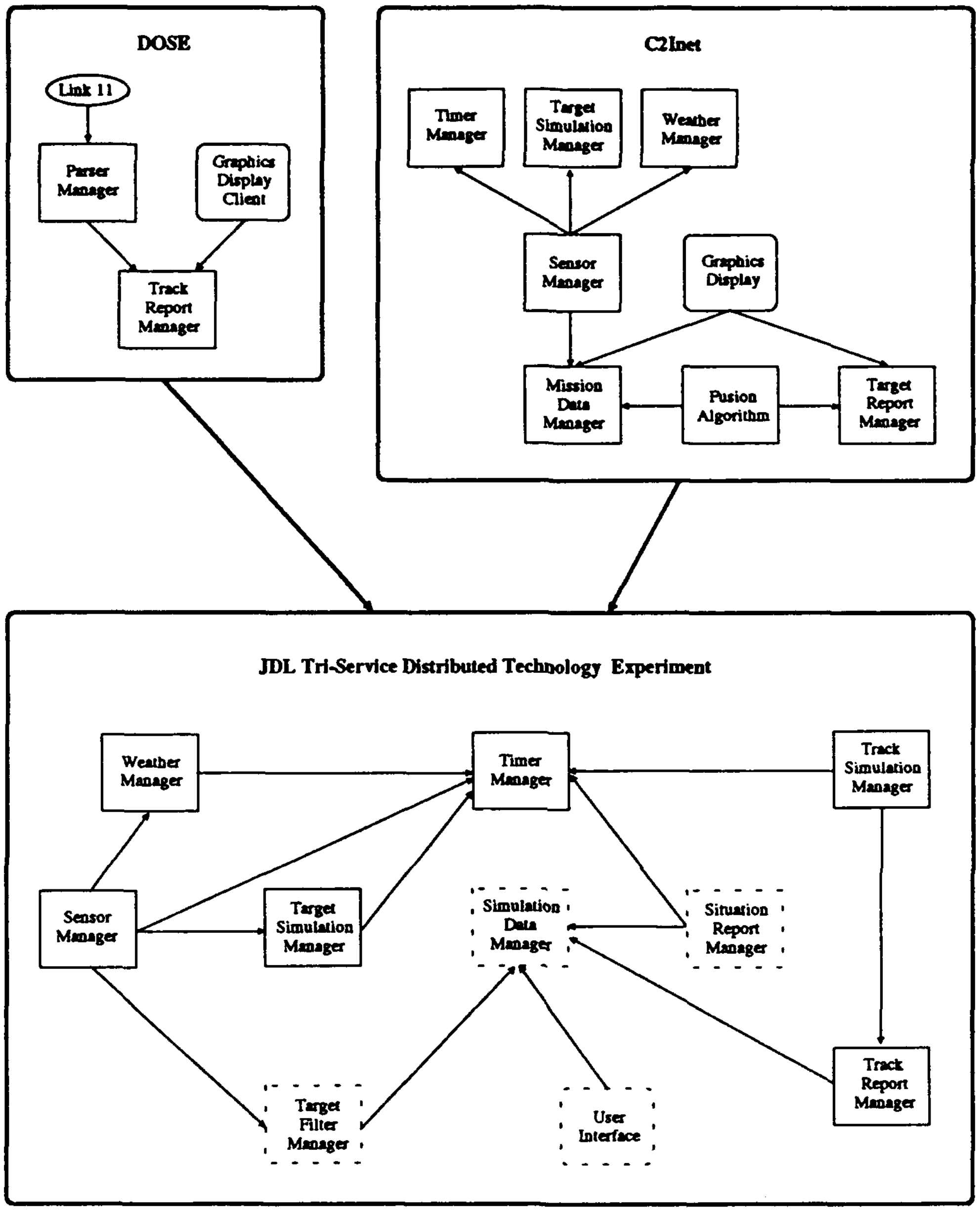


Figure 1. JDL Tri-Service Distributed Technology Experiment.

experiment, the Track Simulation Manager continues to send track information to the Track Report Manager by invoking the operation **TrackUpdate**.

A new scenerio of Navy tracks was created in a data file called *navyfile* in which the tracks appear in the Mediterranean Sea region. For demonstration purposes, the Navy tracks comprise five unique tracks.

Two new operations were added to the Track Simulation Manager as part of the integration of the manager to the experiment: **LinkToDisplay** and **LinkToTimer**. The **LinkToDisplay** operation links the Track Simulation Manager to the JDL Status Display

Manager. The Track Simulation Manager notifies the display manager about its interaction with the Timer Manager and Track Report Manager by invoking the JDL library routines **DisplayStartOp** and **DisplayFinishedOp** (these routines in turn invoke the JDL Status Display Manager operations **StartOp** and **FinishedOp**, respectively). The **LinkToTimer** operation links the Track Simulation Manager to the Timer Manager with a specific timer object. The Track Simulation Manager requests time from the Timer Manager by invoking the operation **GetTime**. The time value is compared with each simulation time field (a new field in the data file) in the *navyfile*. If the simulation time field is equal or less than the time from the Timer Manager, the Track Simulation Manager reads the track information associated with the time.

2.1.2 Track Simulation Manager Operations

The Track Simulation Manager performs 10 operations. The operations **Getdlrp**, **ParseBlock**, and **Setdlrp** are not implemented in the unclassified version of this manager. Four flags control the mode of operation of the Track Simulation Manager: **SetDemoFlag**, **SetDisplayTextFlag**, **SetGraphicsFlag**, and **SetSendtrmFlag**. Each flag can be turned on and off with a default setting as described below. All the Track Simulation Manager operations are invoked via *tropic*.

Getdlrp: This operation can be used to check the value of the Data Link Reference Point (DLRP). The DLRP is a reference point on the globe for all other world coordinates. The operation returns the present DLRP latitude and longitude information (in minutes). **Getdlrp** is not implemented in the unclassified version of the Track Simulation Manager.

LinkToDisplay: This operation notifies the JDL Status Display Manager as to where the Track Simulation Manager was activated (host machine and site location) by invoking the JDL library routine **LogWithDisplay** (which in turn invokes the JDL Status Display Manager operation **HereIAm**).

LinkToTimer: This operation links the Track Simulation Manager to a specified timer object (Timer Manager) for simulation time.

ParseBlock: This operation parses the block of Link 11 data contained in the array of Link 11 messages and attempts to send the parsed ASCII data to a Track Report Manager. Parsed data is sent in blocks to reduce the number of invocations on the Track Report Manager. **ParseBlock** is not implemented in the unclassified version of the manager.

ReadTracks: This operation was created to replace the **ParseBlock** operation in the unclassified version of the Track Simulation Manager (Parser Manager). **ReadTracks** reads already parsed track information from an ASCII data file, *navyfile* (see appendix B.1). The file groups the track information into blocks. Each block contains a varied amount of track information. At the beginning of each block is a number that indicates the number of tracks in that block. Each line in the file represents the data that define the track: track number, latitude of the track (minutes), simulation time (seconds), longitude of the track (minutes), depth or height of the track (feet), latitude of DLRP (minutes), longitude of DLRP (minutes), type of track (S = ship, A = aircraft, U = subsurface ship), category of track (F = friendly, H = hostile, U = unknown), Greenwich mean time, track quality, latitude direction (N = north, S = south), longitude direction (E = east, W = west), course, speed (dm/hr), range from DLRP (nautical miles), and nuclear status (Y = yes, N = no, U = unknown, x = unreported).

SetDemoFlag: In the original DOSE application, this operation was implemented to provide the Parser Manager the capability to avoid timeout errors and overloading the Track Report Manager in a demonstration situation. When the flag is set on, the manager will calculate and set a timeout variable based on a prediction of the time the Track Report Manager will take to process each piece of track information. The Parser Manager (Track Simulation Manager) will sleep according to the timeout variable between invocations on the Track Report Manager. The timeout variable usually defaults to 30 seconds. The flag is set to off by default.

SetDisplayTextFlag: This operation sets the display text flag on or off. When the flag is set on, the Track Simulation Manager will print the parsed messages to a terminal. This flag is usually turned on to debug the Track Simulation Manager and check for correctness of parsing information. This flag is set off by default.

Setdlrp: Due to the nature of the Link 11 data, the user must set the DLRP prior to sending data to the Parser Manager. The parameters of the operation are the latitude and longitude (in minutes) of the new DLRP. This operation is not implemented in the unclassified version of the Track Simulation Manager.

SetGraphicsFlag: The original Parser Manager has a graphics capability (using the SunCore graphics tool) that displays monitoring information about its status and the data flowing through it. This operation sets the graphics flag on or off. This flag should only be turned on when the manager has access to a graphics monitor. Attempting to turn the flag on without a graphics monitor will cause the manager to exit. If the flag has been turned on previously, this operation will clear the monitor of any existing graphical displays. The default setting for the flag is off. This operation is not used by the tri-service experiment.

SetSendtrmFlag: The Track Simulation Manager passes track information to the Track Report Manager for storage. It is not necessary to have the Track Report Manager running for debugging and testing purposes of the Track Simulation Manager. However, if the Track Report Manager is not running, the Track Simulation Manager will attempt to invoke it, resulting in repeated timeouts. **SetSendtrmFlag** allows the user to set the operation flag off, which signals the Track Simulation Manager not to send track information to the Track Report Manager. This flag is set on by default.

2.2 TRACK REPORT MANAGER

In the DOSE application, the Track Report Manager services the Parser Manager and Graphics Display Client. It manages a database of data objects that are replicated with each copy of the Track Report Manager to maintain availability and survivability of the data objects. The replication strategy of version voting is provided by Cronus, which is responsible for maintaining consistency and resolving conflicts between replicated object databases. The Track Report Manager objects are defined to be automatically replicated (the objects are replicated when created).

Each data object is a record of track information sent by the Parser Manager. The representation of the data object is the canonical type, *TrackObject*, which is comprised of two subtypes, *TrackReport* and *TrackReportPlus*. *TrackReport* defines the subset of track information needed to describe the track location and characteristics for the Graphics Display Client to display on the map. *TrackReportPlus* defines additional track information (see appendix A.1).

The track information received by the Track Report Manager is either new or an update to an existing track object. The Track Report Manager searches through its object database using the track number (*trk_num*) as the key. If the track number is not found, an object is created storing the new track information and added to the database. Otherwise, the existing information of the track object with the matched track number is overwritten with the updated information.

The Track Report Manager is written in the C language. It is implemented to run in a UNIX environment.

2.2.1 Integration of Track Report Manager

In the tri-service experiment, the Track Report Manager continues to manage and store Navy track information received from the Track Simulation Manager; however, it no longer services the Graphics Display Client. In the tri-service experiment, the Track Report Manager was adapted to send navy track information to the Simulation Data Manager by invoking the JDL library routines **StoreSimData** and **UpdateSimData** (these routines in turn invoke the Simulation Data Manager operations **StoreData** and **UpdateData**, respectively).

One new operation was added to the Track Report Manager as part of the integration of the manager to the experiment, **LinkToDisplay**, which links the Track Report Manager to the JDL Status Display Manager. The Track Report Manager notifies the JDL Status Display Manager about its interaction with the Track Simulation Manager and Simulation Data Manager by invoking the JDL library routines **DisplayStartOp**, **DisplayFinishedOp**, **DisplayMsgTo**, and **DisplayGotAnswer** (these routines in turn invoke the JDL Status Display Manager operations **StartOp**, **FinishedOp**, **MsgTo**, and **GotAnswer**, respectively). The following section describes the operations available on the Track Report Manager.

2.2.2 Track Report Manager Operations

The Track Report Manager performs 12 operations. Only four of these operations are used in the tri-service experiment: **InitializeDB**, **LinkToDisplay**, **SetTextFlag**, and **TrackUpdate**. The other operations are enhanced software designs based on the DOSE application. The **InitializeDB** operation must be invoked each time the Track Report

Manager is started before any other operation is performed on the manager. **LinkToDisplay** and **SetTextFlag** are invoked for demonstration purposes; thus, they are optional. **TrackUpdate** is automatically invoked by the Track Simulation Manager. All the operations are invoked via *tropic* except the operation **TrackUpdate**, which is invoked by the Track Simulation Manager.

FlushAllTracks: This operation is invoked by the DOSE version of the Graphics Map Client to retrieve all the track information in the Track Report Manager object database. The Track Report Manager does not service the Graphics Map Client in the tri-service experiment; thus, this operation is not used.

GetCurrentTracks: This operation is invoked by the DOSE version of the Graphics Map Client to retrieve the most current set of track information from the Track Report Manager object database that the client has not seen. The Track Report Manager does not service the Graphics Map Client in the tri-service experiment; thus, this operation is not used.

GetNextTrack: This operation was one of the original operations invoked by the DOSE version of the Graphics Map Client to retrieve the track information of the next track number in the queue. This operation is not implemented.

GetObjectDBStatus: To monitor the status of replicated Track Report Managers, a Database Monitor Client (of the DOSE application) invokes the **GetObjectDBStatus** operation, which returns the number of objects managed by each Track Report Manager and the epoch number of the last update of the objects in the database. The Track Report Manager does not service the Database Monitor Client in the tri-service experiment; thus, this operation is not used.

GetTInfo: For debugging purposes, the **GetTInfo** operation returns the information about the tasks currently running.

GetTrack: This operation is invoked by the DOSE version of the Graphics Map Client to retrieve the full track information of a specified track number from the Track Report Manager object database. The Track Report Manager does not service the Graphics Map Client in the tri-service experiment; thus, this operation is not used.

InitializeDB: The Track Report Manager manages two object types, TrackReportData and TrackTableIndex. TrackReportData is the type definition for the track objects that the Track Report Manager manages. TrackTableIndex is the type definition for the TrackTable, which is a list of track entries. Each track entry consists of a unique track number, an object unique identifier (UID) that identifies the track object associated with the track number, and a version number that records the last time the track object was updated. The TrackTable is used as a reference to the track objects maintained by the Track Report Manager's object database. The **InitializeDB** operation removes the data in the object database of type TrackReportData to ensure a clean database when first running the Track Report Manager. The operation also creates a new object for the type TrackTableIndex that stores the TrackTable. The UID of the TrackTable object is stored in the Track Report Manager's generic object as a reference to identify which (nongeneric) object maintains the TrackTable. **InitializeDB** must be invoked before any other operations are invoked on the Track Report Manager.

<u>InitQueue</u>: This operation was one of the original operations invoked by the DOSE version of the Graphics Map Client to initialize a newly allocated queue with its host identifier (ID). This operation is not implemented.

<u>LinkToDisplay</u>: This operation notifies the JDL Status Display Manager as to where the Track Report Manager was activated (host machine and site location) by invoking the JDL library routine **LogWithDisplay** (which in turn invokes the JDL Status Display Manager operation **HereIAm**).

<u>SetMonitorFlag</u>: For demonstration purposes, the Track Report Manager has a graphics capability (using the SunCore graphics tool) that displays the flow of operation invocation from Parser Manager and Graphics Map Client (for the DOSE application). **SetMonitorFlag** allows the user to activate the Track Report Manager graphics by setting the flag to on. The flag is set to off by default. This operation is not used in the tri-service experiment.

<u>SetTextFlag</u>: For debugging purposes, **SetTextFlag** is invoked to provide textual display of the tracks the Track Report Manager received from the Track Simulation Manager for checking the correctness of the information. The flag is set to off by default.

<u>TrackUpdate</u>: The Track Simulation Manager passes an array of track information to the Track Report Manager by invoking the **TrackUpdate** operation. Each track is searched in the TrackTable using the track number as the search key. If the track number is not found in the TrackTable, this operation will create a new object to store the new track information. The track number and the UID of the newly created object are entered in the TrackTable for future reference. If the track number is found in the TrackTable, the object UID associated with the track number is used to locate the track object in the Track Report Manager's object database, in which the new track information overwrites the old information. The version number in the TrackTable associated with this track number is incremented to reflect the update of the track object.

2.3 SENSOR MANAGER

The primary role of the Sensor Manager is to acquire target information from the Target Simulation Manager, Weather Manager, and Timer Manager that is used to generate a series of sensor detections containing real detections and simulated noise. This list is stored in the C2Inet application component Mission Data Manager (see figure 1).

The Sensor Manager objects represent observation platforms that are mobile except when acquiring information. These observation platforms are simulated by associating several data structures with an object. Each object has a schedule that determines the region of space that is observed for some duration of time. These schedules can be stacked into a list so that the sensor object will move from place to place during the simulation. In addition to the schedule, a sensor also contains a parameter that controls its resolution, where a smaller resolution will produce more accurate observations at an increased cost in storage requirements (see appendix A.2).

While a Sensor Manager can contain many sensor objects, only one object can be active (operating) at a time. Once a sensor is operational, it will invoke operations on the Timer Manager (to determine the schedules it should use), the Target Simulation Manager (to determine the targets in its observed region), and the Weather Manager (to add realism to the resolution accuracy). Once target and weather information is received, the sensor maps out a series of real and false detections for a discrete interval of time.

The Sensor Manager is written in the C language. It is implemented to run in a UNIX environment.

2.3.1 Integration of Sensor Manager

In the tri-service experiment, the Sensor Manager was adapted to forward its information to the new Target Filter Manager (where real and false detections are separated) by invoking the operation **FilterDetections**. The sensor also stores the detections in a local constituent operating system (COS) file. This file serves as a backup copy of data should the Target Filter Manager ever fail or be inaccessible. This pattern continues until the sensor reaches the end of its schedule. At the end of the schedule, the sensor is deactivated. The Sensor Manager continues to obtain target, timer, and weather information from the Target Simulation Manager, Timer Manager, and Weather Manager by invoking the operations **TargetsInRegion**, **GetTime**, and **ShowForecastNearLocation**, respectively.

The initialization of the Sensor Manager is activated by the new command file, *initmedsensor* (see appendix B.2). In the tri-service experiment, the sensors are initialized in the Mediteranean Sea region.

Three new operations were added to the Sensor Manager as part of the integration of the manager to the tri-service experiment: **KeepRawData**, **LinkToDisplay**, and **List-Sensors**. The **LinkToDisplay** operation links the Sensor Manager to the JDL Status Display Manager. The Sensor Manager notifies the display manager about its interaction with the Target Filter Manager, Target Simulation Manager, Timer Manager, and Weather Manager by invoking the JDL library routines **DisplayStartOp** and **Display-FinishedOp** (these routines in turn invoke the JDL Status Display Manager operations **StartOp** and **FinishedOp**, respectively). The ListSensors operation is a rename of the C2Inet operation **GetUIDs**. The following section describes the operations available on the Sensor Manager.

2.3.2 Sensor Manager Operations

The Sensor Manager performs 12 operations. All the operations are invoked via *tropic*.

<u>Create</u>: This operation generates a new sensor object given the label (Name), characteristic (SensorType), and resolution (Resolution) of the sensor (see appendix A.2). The new object is catalogued in the default Cronus directory of the Sensor Manager.

The catalog name is a concatenation of the creation host name and the sensor label. The sensor schedule is not defined at creation. In order to adjust the schedule, a separate operation, **MoveTo**, must be invoked. The **Create** operation returns the UID of the newly created object.

Edit: This operation replaces the data in the current sensor object with the operation parameter (Schedule).

GetDetections: This operation returns the full set of detections that a sensor has made while on the current mission. If the sensor is not active, no list is returned.

KeepRawData: This operation initiates the recordkeeping of the raw sensor output on disk. This datafile can be kept for survivability demonstrations on systems where disk space is not an issue, since the file can grow to be many megabytes in size. Once this operation is activated, it is active for the duration of the manager.

LinkToDisplay: This operation notifies the JDL Status Display Manager as to where the Sensor Manager was activated (host machine and site location) by invoking the JDL library routine **LogWithDisplay** (which in turn invokes the JDL Status Display Manager operation **HereIAm**).

LinkToTimer: This operation orients the Sensor Manager onto an individual timer object (TimerObj). Until the timer object is replaced by another operation, the Sensor Manager will attempt to obtain simulation time from the timer object. Until the timer object is set, the Sensor Manager will refuse to operate a sensor.

ListSensors: This operation provides a list of all the sensor objects that are currently resident in the manager object database. The UIDs, as well as the sensor labels and characteristics, are listed in an array form. Each index for each list points to information about the same object. ListSensors returns a list for each of the following: the Cronus identifiers for the objects, the names of the objects, and the characteristics of the objects.

MoveTo: This operation adds an observation frame to the schedule of the sensor. The observation frame is placed at the end of the schedule and is checked to verify that it is correctly time oriented within the schedule.

OperateSensor: This operation selects the target object and activates its schedule. This operation will not complete until either a sufficient error has occurred or the entire schedule has been exhausted; however, once some initial processing has been completed, the manager will send a NULL reply to the invoker. The mode of operation for the sensor is of the following. A sensor will wait until the first observation frame in its schedule becomes active by requesting simulation time updates from the Timer Manager. Once active, it will query the Target Simulation Manager and Weather Manager at the intervals defined by the observation frame. It takes both the target and weather information to create a set of real and imaginary targets that represent the natural efficiency of a real sensor. The real and imaginary targets are placed in a COS file and sent to the Target Filter Manager for processing. Each observation frame creates a new MissionData object and a new COS file. The COS file is not linked into the Cronus file system. At the end of an observation frame, the manager will wait until the next frame becomes active, or the sensor will become deactivated if there are no more observation frames in the sensor.

ReportStatus: This operation returns information regarding the current performance of the manager. The duration of time that the manager has been active (MgrRunTime) is given, along with the currently active sensor object (ActiveSensor). Connection statistics are maintained and displayed with respect to the Timer Manager, Target Simulation Manager, Target Filter Manager, and Weather Manager. These statistics report the number of connection attempts, failures, errors, and timeout values.

SetDetectionThreshold: This operation sets the threshold of the sensor to the operation parameter (Threshold). The threshold determines the level of signal that must be present for the sensor to determine whether a target is going to be reported.

ShowSchedule: This operation returns the entire sensor object (Schedule). Within the object is the sensor schedule, which dictates the regions of space that the sensor is observing during the simulation.

2.4 TARGET SIMULATION MANAGER

The Target Simulation Manager provides data to the Sensor Manager in the form of a list containing target positions defined within the sensor's observed region of space, and the current simulation time. Targets are considered to be any unit of matter that the sensor might detect. Thus, airplanes, tanks, troops, and other items are able to be simulated. Motion of targets is accomplished through the use of a track list that defines a series of path segments. Each path segment is accompanied by a measure of time that is used to determine the individual path segment active at any point in simulated time. The Target Simulation Objects are replicated, with the maximum number of copies set at three. Anyone can read from any Target Simulation Manager, but it takes all copies to update any object. The number of replicated copies may be changed by altering the type definition (*targsim.typedef*) file.

The Target Simulation Manager interacts with the Timer Manager for the current simulation time. Simulation time is used in determining the location of individual targets when the **TargetsInRegion** operation is being performed.

A target is comprised of name and description represented by the canonical type, *TARGETvariable* (see appendix A.3). The Target Simulation Manager is written in the C language. It is implemented to run in a UNIX environment.

2.4.1 Integration of Target Simulation Manager

The Target Simulation Manager maintained its original functionality in the tri-service experiment, providing target detections to the Sensor Manager and obtaining simulation time from the Timer Manager by invoking the operation **GetTime**.

The initialization of the Target Simulation Manager is activated by the new command file, *initmedtargs* (see appendix B.3). In the tri-service experiment, target data are initialized in the Mediteranean Sea region.

One new operation was added to the Target Simulation Manager as part of the integration of the manager to the tri-service experiment, **LinkToDisplay**. The **LinkToDisplay** operation links the Target Simulation Manager to the JDL Status Display Manager. The Target Simulation Manager notifies the display manager about its interaction with the Sensor Manager and Timer Manager by invoking the JDL library routines **DisplayStartOp, DisplayFinishedOp, DisplayMsgTo,** and **DisplayGotAnswer** (these routines in turn invoke the JDL Status Display Manager operations **StartOp, FinishedOp, MsgTo,** and **GotAnswer,** respectively) The following section describes the operations available on the Target Simulation Manager.

2.4.2 Target Simulation Manager Operations

The Target Simulation Manager performs seven operations. All the operations are invoked via *tropic* except the operation **TargetsInRegion**, which is invoked by the Sensor Manager.

Create: This operation creates a target object. It requires the name of the target (Name), the description of the target (TargType), the threat characteristic/intent of the target (Threat), the time the target first appears (StartTime), and the location of the target when it appears (Location). The target forms an initial condition whereby the first entry in its path starts at location (0,0,0) for time zero and continues to location (Location) for time StartTime. The first optional parameter (StartFromOrigin) determines whether the object is visible before the StartTime given in a previous parameter.

DumpSegments: This operation returns the path list that describes the motion of the target. The path list is represented as a series of segments (Tracks) starting from time zero and moving forward to the last segment in the path list. The name (Name), description (TargType), and classification (Class) of the target are also returned.

LinkToDisplay: This operation notifies the JDL Status Display Manager as to where the Target Simulation Manager was activated (host machine and site location) by invoking the JDL library routine **LogWithDisplay** (which in turn invokes the JDL Status Display Manager operation **HereIAm**).

LinkToTimer: This operation allows the manager to lock onto a particular simulation time object. Once done, the manager can successfully get the current simulation time. If the manager needs a time without being linked to a simulation time object, it will use the generic timer object.

MoveTo: This operation adds another segment to the path list that describes the motion of the target. The new segment is formed by taking the last known position in the path list and using it as the starting point of the new segment. The new segment terminates at location (EndLocation) for time (EndTime).

ReportStatus: This operation returns the current position of the object. The Timer Manager is queried for the simulation time, then, the currently active segment is located and the position of the target within the segment is made by straight line approximation.

<u>TargetsInRegion</u>: This operation searches the entire object database looking for targets that fit the search profile defined by the input parameters that define both a region of space (Area) and the time span (StartTime, EndTime) of interest. The operation returns a list that identifies the objects that fit the profile.

2.5 TARGET FILTER MANAGER

The Target Filter Manager receives all detections from the Sensor Manager, filters out false detections, and stores real and semi-real targets within the Simulation Data Manager. A detection is identified as either real or false based on the TargetUID field. Detections that have a valid UID in the TargetUID field are considered to be real targets. Detections that have a NULL TargetUID field are considered to be false targets. Real targets are stored in the Simulation Data Manager. False targets are stored internally and checked against any future false detections. A false target is logged as a semi-real target if it (1) is detected twice within a 50-mile-square region of its original location, (2) has the same TargetDescription, and (3) has the same TargetType. Semi-real targets are stored within the Data Manager as unknown threats. They are updated only if another false detection is found within a 50-mile-square region of its previous location. The Target Filter Manager does not have replicated objects; however, there are multiple instances of the manager operating in the simulation at the same time.

The Target Filter Manager object is represented by the canonical type, *FILTERreport* (see appendix A.4). The manager is written in the C language. It is implemented to run in a UNIX environment.

2.5.1 Integration of Target Filter Manager

The Target Filter Manager is a new JDL component that interacts with the Sensor Manager and Simulation Data Manager. The target information received from the Sensor Manager is sent to the Simulation Data Manager by invoking the JDL library routines **StoreSimData** and **UpdateSimData** (these routines in turn invoke the Simulation Data Manager operations **StoreData** and **UpdateData**, respectively).

The **LinkToDisplay** operation links the Target Filter Manager to the JDL Status Display Manager. The Target Filter Manager notifies the display manager about its interaction with the Sensor Manager and Simulation Data Manager by invoking the JDL library routines **DisplayStartOp**, **DisplayFinishedOp**, **DisplayMsgTo**, and **DisplayGotAnswer** (these routines in turn invoke the JDL Status Display Manager operations **StartOp**, **FinishedOp**, **MsgTo**, and **GotAnswer**, respectively). The following section describes the operations available on the Target Filter Manager.

2.5.2 Target Filter Manager Operations

The Target Filter Manager performs three operations. All the operations are invoked via *tropic* except the operation **FilterDetections**, which is invoked by the Sensor Manager.

FilterDetections: This operation takes the incoming array of detections and transforms them into a set of real and semi-real targets that are stored in the Data Manager.

IdentifiedTarget: This operation is sent to all TargetFilter managers to notify them that a target is being tracked and maintained by another TargetFilter manager. This information is used when one manager takes over the targets of another manager because of failure.

LinkToDisplay: This operation notifies the JDL Status Display Manager as to where the Target Filter Manager was activated (host machine and site location) by invoking the JDL library routine **LogWithDisplay** (which in turn invokes the JDL Status Display Manager operation **HereIAm**).

RemoveObjectHistory: This operation will clear any memory that the Target Filter Manager has about any previous targets that it has seen.

ReportStatus: This operation returns information regarding the total number of detections processed; percentage of real, semi-real, and fake targets; the total number of operations processed; which sensor sent the operations; and how many operations were sent per sensor.

2.6 WEATHER MANAGER

The Weather Manager provides data to the Sensor Manager in the form of a weather forecast. These forecasts can either be created before or during the simulation. The Weather Manager interacts with the Timer Manager for the current simulation time.

The Weather Manager manages two objects, WeatherData Object and WeatherForecast Object. The WeatherData Object provides the anchor node in the JDLSimulation Type hierarchy for the weather forecast (jdlfcast) and weather report (jdlwxrpt) objects. The WeatherForecast Object contains basic weather information for a map location as well as date information to assist in determining the importance and availability of weather information. While users may create weather forecast objects, all operations are generic in nature; thus, no user ever has direct access to an individual forecast object (see appendix A.5).

The Weather Manager is written in the C language. It is implemented to run in a UNIX environment.

2.6.1 Integration of Weather Manager

The Weather Manager maintained its original functionality in the tri-service experiment, providing weather information to the Sensor Manager and obtaining simulation time from the Timer Manager by invoking the operation **GetTime**.

One new operation was added to the Weather Manager as part of the integration of the manager to the tri-service experiment, **LinkToDisplay**. The **LinkToDisplay** operation links the Weather Manager to the JDL Status Display Manager. The Weather Manager notifies the display manager about its interaction with the Sensor Manager and Timer Manager by invoking the JDL routines **DisplayStartOp, DisplayFinishedOp, DisplayMsgTo,** and **DisplayGotAnswer** (these routines in turn invoke the JDL Status Display Manager operations **StartOp, FinishedOp, MsgTo,** and **GotAnswer,** respectively). The following section describes the operations available on the Weather Manager.

2.6.2 Weather Manager Operations

The Weather Manager performs six operations. All the operations are invoked via *tropic* except the operation **ShowForecastNearLocation,** which is invoked by the Sensor Manager.

Create: This operation creates a weather forecast object instance.

LinkToDisplay: This operation notifies the JDL Status Display Manager as to where the Weather Manager was activated (host machine and site location) by invoking the JDL library routine **LogWithDisplay** (which in turn invokes the JDL Status Display Manager operation **HereIAm**).

LinkToTimer: This operation allows the manager to lock onto a particular simulation timer object. Once done, the manager can successfully get the current simulation time. If the manager needs a time without being linked to a simulation timer object, it will use the generic timer object.

RemoveOutdatedForecasts: This operation forces the manager to search through its object database in search of expired forecasts. The optional parameter (RemoveFrom) defines a reference time to be used in the elimination process. If RemoveFrom is specified, the manager will remove all outdated forecasts from the beginning of time to the time specified. If RemoveFrom is not specified, the manager will get the current simulation time and remove all forecasts from the beginning of time to the current simulation time. This operation returns the total number of forecasts that were deleted.

ReportStatus: This operation returns a message regarding the Uptime of the manager and some statistical data indicating the managers success in communicating with the Timer Manager.

ShowForecastNearLocation: This operation forces the manager to search through its object database in search of a forecast that meets the requirements of the two parameters (neither of which are optional). Location defines a coordinate point on the map (two dimensional). Time defines the simulation interval time of interest. This operation will return the forecast closest to or containing the point of interest. If there are multiple entries, the forecast last accessed by the manager will be returned.

2.7 TIMER MANAGER

In the C2Inet application, the primary function of the Timer Manager was to provide simulation time for the Sensor Manager, Target Simulation Manager, and Weather Manager. The Timer Object contains timing information for various simulations that may be running with Cronus. Each individual timer object represents a simulation clock that may be manipulated in a wide range of operations. In general, simulation time can be viewed as the interval of time that has elapsed since the start of the timer object. The generic object serves as a global timekeeper, maintaining the current constituent system time.

Simulation timers (individual timer objects) can be stopped, started, and reset to their initial values. In addition, simulation timers can scale themselves to produce simulation clocks that run faster or slower than the constituent system time clock. Thus, when applications use the timer objects to drive application events, the simulation time provided by the timer objects can speed up or slow down the application for demonstration, observation, or debugging.

Each timer object represents a different simulation clock. The simulation clock database is represented by the canonical type, TIMERobject (see appendix A.6).

The Timer Manager is written in the C language. It is implemented to run in a UNIX environment.

2.7.1 Integration of Timer Manager

In the tri-service experiment, the Timer Manager continues to provide simulation time to the Sensor Manager, Target Simulation Manager, and Weather Manager; in addition, it provides simulation time to the Track Simulation Manager, Situation Report Manager, and JDL Status Display Manager.

One new operation was added to the Timer Manager as part of the integration of the manager to the tri-service experiment, **LinkToDisplay**. The **LinkToDisplay** operation links the Timer Manager to the JDL Status Display Manager. The Timer Manager notifies the display manager about its interaction with the aforementioned managers by invoking the JDL routines **DisplayStartOp**, **DisplayFinishedOp**, **DisplayMsgTo**, and **DisplayGotAnswer** (these routines in turn invoke the JDL Status Display Manager operations **StartOp**, **FinishedOp**, **MsgTo**, and **GotAnswer**, respectively). The following section describes the operations available on the Timer Manager.

2.7.2 Timer Manager Operations

The Timer Manager performs 20 operations. All the operations are invoked via *tropic* except the operation **GetTime**, which is invoked by the Track Simulation Manager, Sensor Manager, Target Simulation Manager, Weather Manager, Situation Report Manager, and JDL Status Display Manager for simulation time.

Create: This operation creates a simulation timer object instance.

GetHostTime: This operation returns the current constituent operating system date and time.

GetTime (generic): This generic operation returns the interval of time elapsed since the manager has started.

GetTime: This operation returns the interval of time elapsed since the timer object was started.

LinkToDisplay: This operation notifies the JDL Status Display Manager as to where the Timer Manager was activated (host machine and site location) by invoking the JDL library routine **LogWithDisplay** (which in turn invokes the JDL Status Display Manager operation **HereIAm**).

ReportStatus (generic): This generic operation returns information on the current working parameters of the generic timer object.

ReportStatus: This operation returns relevant information on the performance (the current simulation time (Elapsed), the current constituent system time (GlobalTime)), and current state (Attributes) of the timer object.

Reset: This operation sets the measurement values so that the effective time returned by the **GetTime** operation is the ResetValue (initially zero). Reset will automatically stop the timer object if it is running. After a **Reset** operation, the affected timer object must be manually started.

SetQuality: This operation is currently not implemented. When the Timer Manager becomes replicated, this operation will serve to set the quality factor that determines the best manager suited to synchronize the other replicated copies.

SetRate: This operation alters the speed of the simulation time clock. Default values of 1 for both the multiplier and divisor functions allow the simulation clock to count in step with the natural progression of time. If **SetRate** is called without parameters, the current multiplier and divisor rates are reset to the default value of 1. If the Rate-Multiplier parameter is supplied, the simulation clock will quicken by the factor of the RateMultiplier. If the RateDivisor parameter is supplied, the simulation clock will slow by the factor of the RateDivisor. The simulation clock will scale its counting to accommodate RateMultiplier and RateDivisor values that are simultaneously nondefault.

SetResetValue: This operation sets the ResetValue of the timer object to the date parameter (Time). The ResetValue is used to determine the initial starting value of the timer object after a **Reset** operation has been executed. The timer object will count forward from ResetValue when the timer object is started after a reset.

SetStepIncrement: This operation sets the granularity of time that passes within a single cycle of the simulation clock. The StepIncrement parameter establishes the amount of time (either seconds, minutes, or hours) upon which the clock iterates.

SetStopTime: This operation establishes a time when the simulation clock will automatically terminate its timekeeping function. Once set, the ony way to cancel this function is a **Reset** operation on the timer object. The StopTime parameter will set the time for the simulation clock to automatically stop.

SetSynchInterval: This operation is currently not implemented. When the Timer Manager becomes replicated, this operation will serve to set the update interval for keeping copies of the generic timer object base-time measurement synchronized.

SetTime (generic): This generic operation sets the base-time measurement to the date parameter (Time). If Time is not present, the base-time measurement will be set to the current constituent host time. The base-time measurement is used by the nongeneric timer objects as a reference point for calculating their individual simulation times.

SetTime: This operation sets the reference-time measurement to the date parameter (Time). If Time is not present, the reference-time measurement will be set to the base-time measurement of the generic object. The reference-time measurement is used by the timer objects in calculating their individual simulation times.

Start: This operation signals the simulation clock to begin. The simulation clock resumes counting from the time at which it was stopped. The only exception to this rule occurs if the simulation clock has been reset. If reset, the simulation clock starts from the default (ResetValue) position. The StopTime parameter, if supplied, will set the time for the simulation clock to automatically stop.

Step: This operation will, by default, increment the simulation clock by one iterative value. The Count parameter, if supplied, will set the number of cycles that the clock can continue without stopping. Once the count has been exhausted, the simulation clock will stop. Also, the StepIncrement parameter, if supplied, will establish the amount of time that passes in a single cycle of the clock. It is important to note that the StepIncrement parameter is valid beyond this step iteration. The StepIncrement that the clock uses in normal timekeeping is CHANGED. Upon receipt of the Step operation, the simulation clock is stopped, the step interval is computed, the clock is set to stop at the computed step interval, and the clock is started.

Stop: This operation signals the simulation clock to halt. The simulation clock will only resume counting with the issue of a Start operation.

Synchronize: This operation is currently not implemented. When the Timer Manager becomes replicated, this operation will serve to update the copies of the generic timer object base-time measurement.

2.8 SITUATION REPORT MANAGER

The Situation Report Manager simulates information on ground-based forces, both friendly and hostile. The information to be managed is that which would be useful to an Army user in a Division-level command post. This information is divided into unit-level status reports known as Situation Reports. The Situation Report Manager object is represented by the canonical type, *Situation_Report_Data* (see appendix A.7).

Similar to the Track Simulation Manager, the Situation Report Manager obtains its data from a file, *armyfile*. The Situation Report Manager invokes the Timer Manager for time and checks the *armyfile* to see if it is time to read the Army data. Similar to the Track Report Manager, the Situation Report Manager stores the data it retrieves from the *armyfile* into its own object database. It passes the Army data to the Simulation Data Manager for storage.

When associative access of the object database (accessing Cronus objects by content rather than by identifier) was available, the Situation Report Manager was reimplemented with the capability. The SituationReport type was redefined as a subtype of *CronusQuery* rather than *Object*. The operation **LocateSitRep** was rewritten to use the **SelectMatchingObjects** library routine.

2.8.1 Symbolic Version of Situation Report Manager

The Situation Report Manager was initially implemented in the C language. To gain experience with the Symbolic implementation of Cronus, a Lisp version of the Situation Manager was designed. These two versions of the Situation Report Manager enhance the heterogenity environment for the tri-service experiment.

As part of the design of the Lisp version of the Situation Report Manager and as part of a rethinking of its overall functionality, two new operations were added: **Create** and **Update**. **Create** is a generic operation used to create new SitRep objects, and the **Update** operation updates existing SitRep objects. The benefit of implementing these operations is to give the Situation Report Manager a more general form, and thus, provide the manager the capability to survive outside of the tri-service experiment.

2.8.2 Integration of Situation Report Manager

The Situation Report Manager is a new JDL component that interacts with the Timer Manager and Simulation Data Manager. The Situation Report Manager obtains simulation time from the Timer Manager by invoking the operation **GetTime**. Situation Reports are sent to the Simulation Data Manager for storage by invoking the JDL library routines **StoreSimData** and **UpdateSimData** (these routines in turn invoke the Simulation Data Manager operations **StoreData** and **UpdateData**, respectively).

The **LinkToDisplay** operation links the Situation Report Manager to the JDL Status Display Manager. The Situation Report Manager notifies the display manager about its interaction with the Timer Manager and Simulation Data Manager by invoking the JDL library routines **DisplayStartOp** and **DisplayFinishedOp** (these routines in turn invoke the JDL Status Display Manager operations **StartOp** and **FinishedOp**, respectively). The following section describes the operations available on the Situation Report Manager.

2.8.3 Situation Report Manager Operations

The Situation Report Manager performs nine operations. All the operations are invoked via *tropic*.

Create: This operation performs the creation of Situation Report objects containing the specified information.

LinkToDisplay: This operation notifies the JDL Status Display Manager as to where the Situation Report Manager was activated (host machine and site location) by invoking the JDL library routine **LogWithDisplay** (which in turn invokes the JDL Status Display Manager operation **HereIAm**).

LinkToTimer: This operation allows links the manager to a simulation timer. Until the timer object is set, the manager will refuse to read Situation Reports.

LocateSitRep: This operation locates the Situation Report object for the specified unit.

ReadSitReps: This operation the manager to read situation reports from a data file, *armyfile* (see appendix B.2).

ReportStatus: This operation returns information on the current status of the Situation Report Manager.

SetDisplayTextFlag: This operation sets the display text flag on or off. This flag is set to off by default. When the flag is set on, the Situation Report Manager will print the incoming reports to a terminal. This flag is usually set on to debug the Situation Report Manager.

SetSendDMFlag: This operation sets the send to Data Manager flag on or off. This flag is set on by default. When the flag is set on, the Situation Report Manager will attempt to send the status reports to the Data Manager. This flag is usually turned off for debugging the Situation Report Manager. This is especially useful when the Data Manager is not running and the attempt to invoke it will result in repeated timeouts.

Update: This operation allows the update of the information in a Situation Report.

2.9 SIMULATION DATA MANAGER

The Simulation Data Manager was designed to manage data from the joint services (Navy, Air Force, and Army). The Track Report Manager, Target Filter Manager, and Situation Report Manager use the Simulation Data Manager to report data in their own application-unique forms. In the initial phase design of the Simulation Data Manager, the tri-service data is stored in the manager's object database. The structure of the basic data object is represented by the canonical type, *SIMdata* (see appendix A.8). The data stored and managed by the Simulation Data Manager is displayed on the User Interface.

Cronus Release 1.5 provided two new associative access capabilities: query processing and relational database support. The Simulation Data Manager was reimplemented to provide both capabilities.

2.9.1 Simulation Data Manager—Query Processing

The query processing capability allows clients to invoke operations that identify collections of objects by attribute instead of by their Cronus-unique identifier. Queries are constructed using a subset of the standard Structured Query Language (SQL) used by many relational database systems. These SQL-like query capabilities allow the client program to formulate adhoc queries and send them to the appropriate object manager for processing.

The operations **QueryForData** and **QueryForObjects** are implemented to allow the User Interface to acquire data from the Simulation Data Manager by using query processing.

2.9.2 Simulation Data Manager—Relational Database Support

In previous releases of Cronus, the one mechanism available for storage and retrieval of Cronus objects was the object database. In Cronus Release 1.5, the number of data storage and retrieval options have been expanded to provide direct support for accessing relational database management systems (DBMS) through Cronus. A set of new Cronus managers that encapsulate the various commerical relational databases was implemented to support this capability. The new SQLDefs, DBDefs, and DBSession object types define and implement the set of operations that are available across all the supported relational databases. For each database supported, an object type is defined that is a subtype of DBDefs and of DBSession. These subtypes implement a small number of database-specific customizations that accommodate differences between the various relational database products. The currently implemented relational databases are Informix, Oracle, and Sybase. Other databases may be added later after subtypes are defined and implimented for them.

The operations **QueryForData** and **QueryForObjects** are implemented to allow the User Interface to acquire data from the Simulation Data Manager via Informix, Oracle, or Sybase. These operations replace the operations **RetrieveData** and **RetrieveObjects**. The operations **StoreData** and **UpdateData** were reimplemented to store and update data in a relational database (Informix, Oracle, or Sybase).

2.9.3 Integration of Simulation Data Manager

The Simulation Data Manager is a new JDL component that interacts with the Track Report Manager, Target Filter Manager, Situation Report Manager, and User Interface.

The Simulation Data Manager receives data as objects from the Target Filter, Situation Report, and Track Report managers. Once the objects are received, the data (common to all three services) are broken out and stored in a table in the relational database. The service-specific data are then stored in another table in the relational database. Updates are handled in a similar manner to the store. When information is needed from the database (usually by the User Interface), the operations **QueryForData** and **QueryForObjects** are called. In both of these operations, a SQL query is sent through Cronus to the relational database, and the results are returned to the operation in SelfDescribingTuples (a Cronus structure). The **QueryForData** operation returns the SelfDescribingTuples to the query requester, who then must unpack the SelfDescribingTuples. The operation **QueryForObjects** is used for queries on the common data because this operation unpacks the SelfDescribingTuples it receives and packs an array with the data. The **QueryForObjects** operation was created so that the User Interface would not have to be changed.

The **LinkToDisplay** operation links the Simulation Data Manager to the JDL Status Display Manager. The Simulation Data Manager notifies the display manager about its interaction with the aforementioned managers and User Interface by invoking the JDL library routines **DisplayStartOp**, **DisplayFinishedOp**, **DisplayMsgTo**, and

DisplayGotAnswer (these routines in turn invoke the JDL Status Display Manager operations **StartOp, FinishedOp, MsgTo,** and **GotAnswer** respectively). The following section describes the operations available on the Simulation Manager.

2.9.4 Simulation Data Manager Operations

The Data Manager performs 12 operations. The operations **QueryForData, QueryForObjects, RetrieveData,** and **RetrieveObjects** are invoked by the User Interface for simulation data to be displayed. The operations **StoreData** and **UpdateData** are invoked by the Track Report Manager, Target Filter Manager, and Situation Report Manager to store and update Navy, Air Force, and Army data being stored by the Simulation Data Manager. All the other operations are invoked via *tropic*.

CloseDatabase: This operation closes the database session.

DeleteAllInDatabase: This operation deletes all the records in the database.

LinkToDatabase: This operation attaches the Data Manager to one of three databases: Informix, Oracle, or Sybase. The manager will not operate without the invocation of this operation. LinkToDatabase will return an error code if it cannot successfully attach and open the database specified.

LinkToDisplay: This operation notifies the JDL Status Display Manager as to where the Simulation Data Manager was activated (host machine and site location) by invoking the JDL library routine **LogWithDisplay** (which in turn invokes the JDL Status Display Manager operation **HereIAm**).

QueryForData: This operation returns a self-describing tuple that matches the input parameters. The type of database (Informix, Oracle, Sybase) is returned for use in compatibility issues.

QueryForObjects: This operation makes calls to the database (table datainfo) and returns an array filled with the simulation data that matches the input parameters.

Remove: This operation removes a mission object from the database. The operation is not being imported from the generic RemoveOperation because there are internal storage values that must be released when an object is removed. This is a skeleton operation, having been made obsolete by the relational database version.

ReportStatus: This operation returns the database that is connected to the Data Manager.

RetrieveData: This operation returns an array filled with the simulation data that matches the input parameters. This is a skeleton operation, having been replaced by **QueryForData**.

RetrieveObjects: This operation returns an array filled with the simulation data that match the input parameters. This is a skeleton operation, having been replaced by **QueryForObjects**.

StoreData: The operation **StoreData** creates an empty object and places it in the Tri-Service Data Manager Object database. The UID is used for the key fields in the tables of the relation database. The UID, latitude, longitude, altitude, type, source, threat, category, are incorporated into a SQL Insert statement. The SQL statement is the sent to the relational database. After the general information is stored, the sending manager type is deciphered. The octetbuffer is converted to the original type (such as TrackObject, SituationReport, DECTIONdata). From the type, the data are taken to form the SQL Insert Statement. The SQL Insert Statement is sent to the relational database. **StoreData** returns the UID of the new mission object.

UpdateData: The operation **UpdateData** receives updated data from the managers, using the UID as a key for the data. SQL UPDATE statements are sent to the database with the new information. The UID is then returned to the calling procedure.

2.10 JDL STATUS DISPLAY MANAGER

Separation of components is an inherent property of distributed systems. The JDL Status Display Manager was designed to display the distribution of the experiment application components among the three sites. The manager provides a view into the layout of the experiment by showing location, interaction pattern, and elementary state information (e.g., busy, input-queue-entry, output-queue-entry) for each component.

2.10.1 Diamondtool Versions of JDL Status Display Manager

The JDL Status Display Manager was implemented by using the **Diamond User Interface Toolkit**, developed by BBN. The toolkit is a collection of tools and subroutines that permit a programmer to use user-interface primitives such as menus, scroll-bars, and dialog boxes. It also responds to window-system events such as closing the window into an icon, partially concealing (or unconcealing a window), etc.

In the first design phase of the manager, each component of the experiment has an individual box on the display. Each box is color coded to match the host within which the experiment manager resides. Input/output (I/O) boxes represent message activity between components of the simulation. The JDL Status Display Manager was only able to display one instance of each experiment managers and client (figure 2).

A second-phase design of the JDL Status Display Manager provides display of multiple instances (replication and duplication) of managers and clients. The component Target Filter Manager was added to the display (Target Filter Manager was not available during the first-phase design of the JDL Status Display Manager). There is also a display box showing "SimulationProgress" (figure 3).

2.10.2 Motif Version of JDL Status Display Manager

The Diamond User Interface Toolkit is an unsupported BBN product, which prompted a parallel design of the JDL Status Display Manager with implementation based on X Window system and Motif Widget set. Motif is currently the most popular graphical user interface style for the X Window system (which is rapidly becoming an industry standard). The use of Motif in the tri-service experiment will align the JDL

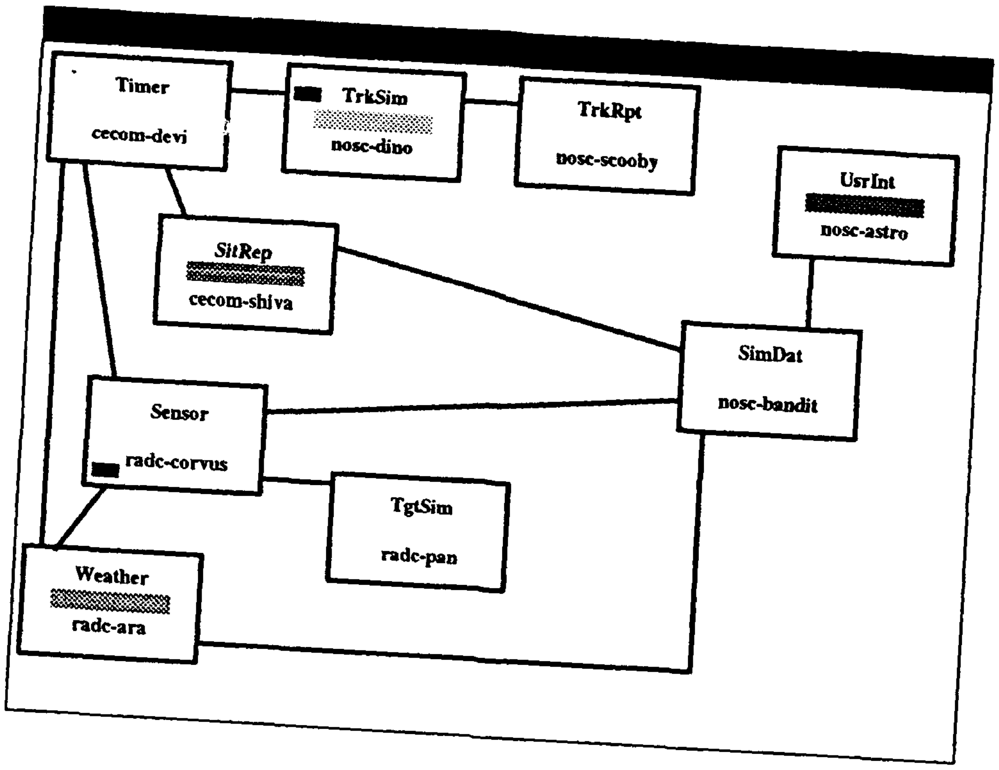


Figure 2. Phase I Diamondtool JDL Status Display Manager.

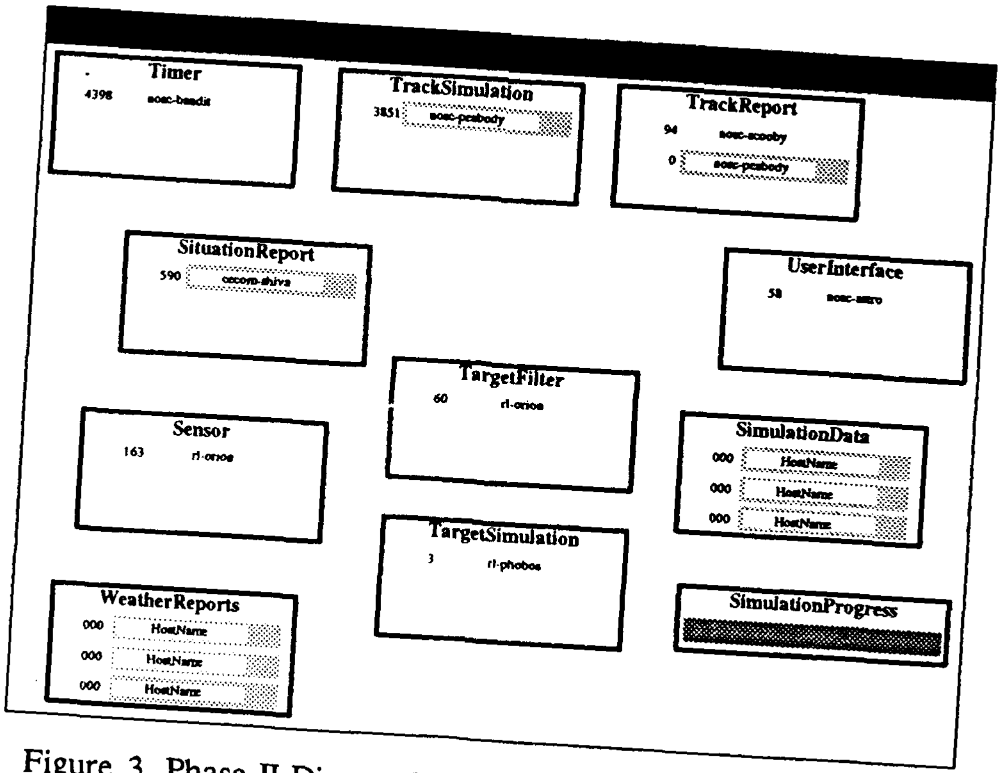


Figure 3. Phase II Diamondtool JDL Status Display Manager.

tri-service with emerging industry standards as well as major programs (NTCS-Afloat) at NRaD.

The Motif version of the JDL Status Display Manager communicates to a separate process issuing drawing commands as a result of progressions in the simulation. The JDL Status Display Manager and its associated X/Motif application evolved into two executables because of performance considerations in having a Cronus manager handle its usual work as well as another X task.

In its current state of development, the JDL Status Display Manger receives simulation progress updates from the other experiment managers and translates these updates into drawing commands to the X/Motif display. Replication of managers is shown more easily in the Motif version. The display can be enhanced much more easily to include new options such as requesting information about a particular manager on a particular host and popping up a window that contains the results (figure 4).

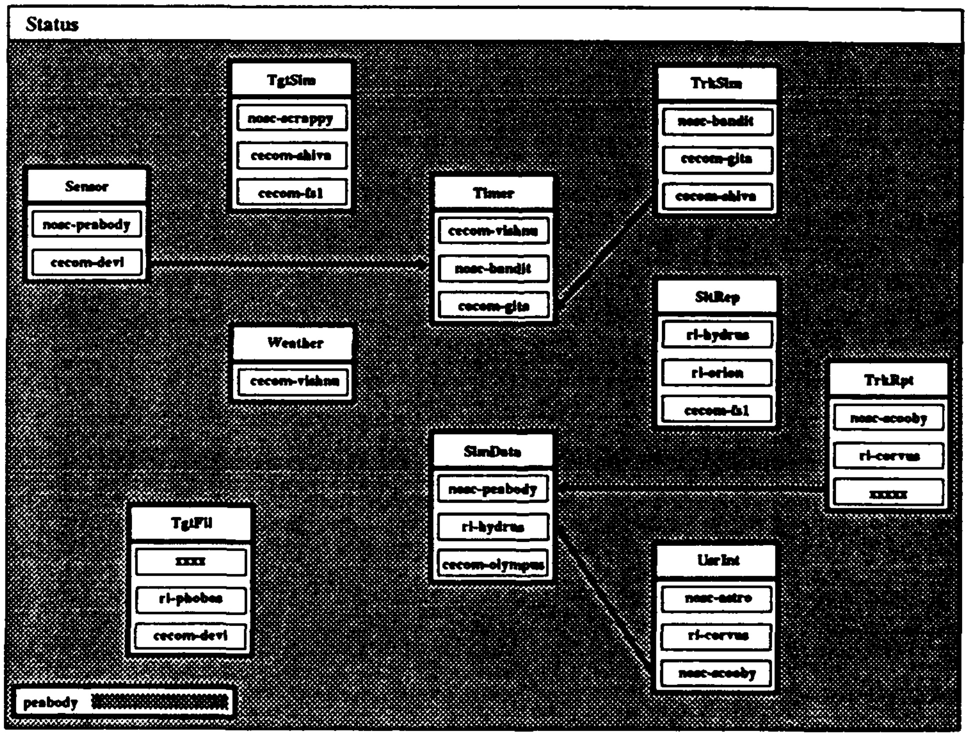


Figure 4. Motif version of JDL Status Display Manager.

2.10.3 JDL Status Display Manager Operations

The JDL Status Display Manager performs seven operations. The operations **FinishedOp, GotAnswer, HereIAm, MsgTo,** and **StartOp** are invoked by the experiment managers and User Interface to register their location and activities with the JDL Status Display Manager. The operations **LinkToTimer** and **Reset** are invoked via *tropic*.

<u>FinishedOp</u>: This operation is invoked by a simulation component that has finished processing an operation request.

<u>GotAnswer</u>: This operation is invoked by a simulation component that has received an answer to its operation request.

<u>HereIAm</u>: This operation registers the individual simulation components with the display manager. It appropriately clears all of the I/O boxes and sets the manager display box to the proper color.

<u>LinkToTimer</u>: This operation attaches the Status Display Manager to a timer object for acquiring simulation time.

<u>MsgTo</u>: This operation is invoked by a simulation component that is invoking an operation on another simulation component.

<u>Reset</u>: This operation resets the display to an initial state (with no colors).

<u>StartOp</u>: This operation is invoked by a simulation component that is processing an operation request.

2.11 USER INTERFACE

In the DOSE application, the Graphics Display Client is a worldwide, dynamically scaled map package. The software uses *SunCore*, a graphics package for Sun workstations, to construct the map from a digital map database. The Graphics Display Client invokes the Track Report Manager for Navy data to be displayed on a world map.

In the tri-service experiment, the Graphics Display Client was adapted as the User Interface. The User Interface was modified and enhanced with the capability to display Navy, Air Force, and Army information.

2.11.1 Integration of User Interface

In the tri-service experiment, the User Interface obtains Navy, Air Force, and Army information from the Simulation Data Manager by invoking the operations **QueryForData** and **QueryForObjects** (query processing version of the Simulation Data Manager) and **RetrieveData** and **RetrieveObjects** (relational database version of the Simulation Data Manager). The Navy, Air Force, and Army data are displayed symbolically as unclassified icons in the Mediteranean Sea region of the world.

The User Interface notifies the JDL Status Display Manager as to where it was activated (host machine and site location) by invoking the JDL library routine

LogWithDisplay operation (which in turn invokes the JDL Status Display Manager operation **HereIAm**).

2.11.2 SunCore Version of User Interface

The SunCore version of the User Interface is a single window display of a world map with button panels to control the display of the map and the type of information obtained from the Simulation Data Manager. The User Interface is mouse controlled.

When the User Interface is first activated, the name of the experiment and the sites participating in the experiment are displayed on the monitor screen. Clicking on the left button of the mouse will display a menu choice of automatic data update mode or manual data update mode. In automatic update mode, the User Interface will automatically invoke the Simulation Data Manager for new data update every 30 seconds. In manual update mode, the user controls when to request new data update for display. After the user chooses the type of update mode by clicking the mouse icon on the appropriate choice, a menu choice of different regions of the world (e.g., Mediterranean, Asia, etc.) is displayed. Again the user chooses the map region by clicking the mouse icon on the appropriate choice. The chosen map region is then displayed on the screen along with display control panels, data icon information, and map information (figure 5). The tri-service experiment scenario takes place in the Mediterranean Sea region.

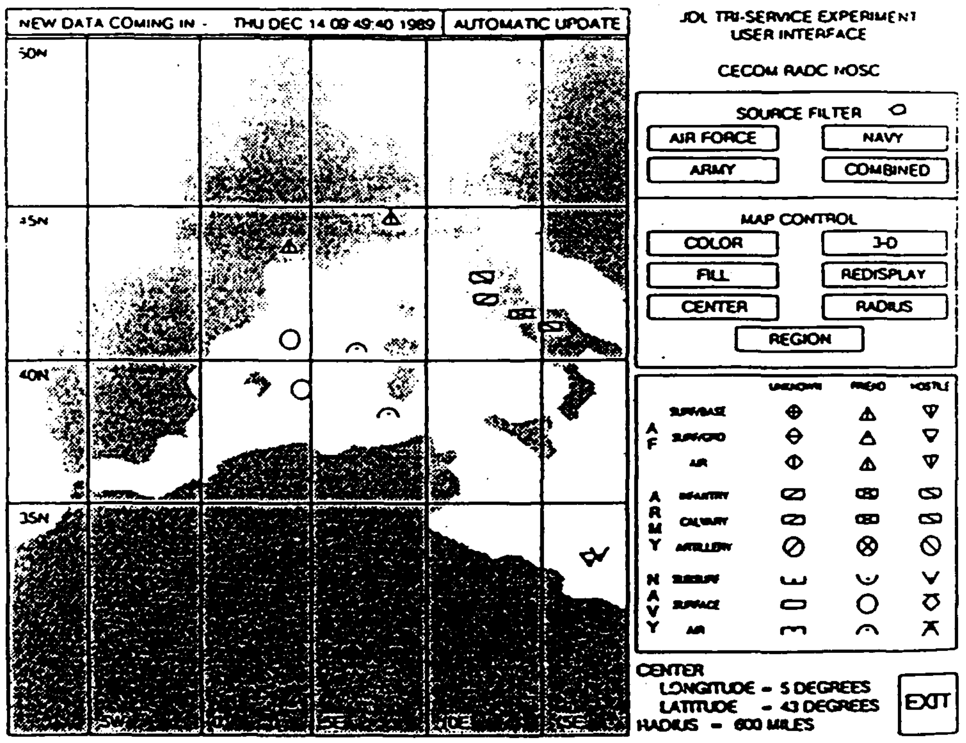


Figure 5. SunCore version of User Interface graphical display.

There are two panels at the top of the map. The left panel notifies the user as to when new data have arrived. The right panel indicates the type of update mode that is currently active. The user can click the mouse icon on the panel and it will toggle between automatic data update mode and manual data update mode.

There are two main control panels on the right side of the map: *Source Filter* and *Map Control*. The *Source Filter* panel allows the user to choose the type of data to be obtained from the Simulation Data Manager and displayed on the map: only Air Force data, only Army data, only Navy data, or tri-service data. Clicking the mouse icon on any of the buttons will display a subpanel, *Display Capabilities*. This subpanel provides the user different means of displaying the specified service(s) data: *Status* (number of and different types of data icons on the map), *Select* (filter data to be displayed on the map according to threat identification and classification types), *Weather* (textual report of the weather condition in the region), *Identify* (textual report on the position, classification, etc. of a selected icon on the map), *New Data* (invokes the Simulation Data Manager for the most recent update of new data), and *Refresh* (refreshes the data and map in case distortion and inconsistency of data occurs).

Map Control allows the user to manipulate the map display. The user can change the color of the land (*Color*), fill or outline the land (*Fill*), change the center of the map (*Center*), tilt the map (*3-D* button for the purpose of displaying air and subsurface data), redisplay the full screen (*Redisplay*), change the radius for panning and zooming purposes (*Radius*), and change the region of the world (*Region*).

2.11.3 X Window Version of User Interface

GOTS is an automated command, control, communications, and intelligence (C3I) system that has been designed for tactical situation assessment. Its function is to receive, manage, process, and display data that are reported on objects (ships, aircraft, etc.) being tracked by various sensor systems. A portion of GOTS, the optional GOTS environment, is being used by the JDL Tri-Service Experiment to develop a new User Interface. The Map Server and Open Software Foundation (OSF)/Motif are the pieces of the GOTS alternate environment being used in developing a tactical User Interface.

The Map Server generates all maps for applications connected to the Map Server. Multiple Map Windows are supported. The Map Server's primary role is to provide application programs with the ability to plot a variety of graphics objects onto the display through a simple library of C or Ada functions. Once plotted, these objects will be remembered and redisplayed whenever the map is repainted. The Map Server provides a menu ("Map Options") that allows the user to manipulate the map display. For example, the user has several centering and zooming options as well as the option of using an orthographic projection rather than a Mercator projection. In essence, much of the foundation for a good user interface is provided by the Map Server and application developers need only add application-specific code.

In its current state of development, the JDL User Interface requests a map from the Map Server which it then reparents to its hierarchy of user interface components

(Widgets). Map events and Widget events are then forwarded to the application in the same fashion from the X server that is taking care of the display. Currently, the User Interface is able to manage the objects on the screen (hi-liting, deleting, hiding) in terms of single objects (one icon on the screen) or in groups of objects (many icons of the same type). It also has the ability to query (via the Simulation Data Manager) a database for information about a specific icon on the display. A query string can also be constructed using a menu to instruct the Simulation Data Manager to filter out certain types of data it normally would forward to the User Interface.

The X version of the User Interface will provide a better-looking, easier-to-use interface to the tri-service experiment as well as use emerging industry standards (X, OSF/Motif) (figure 6).

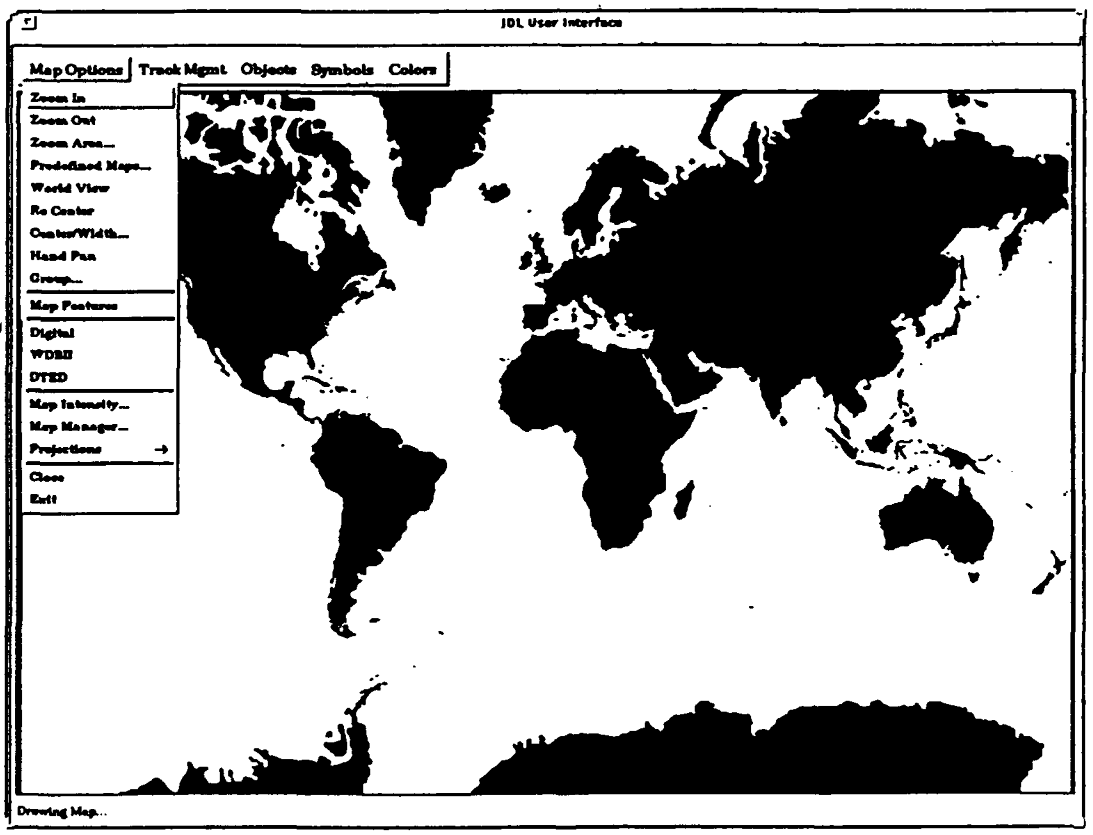


Figure 6. X version of the User Interface graphical display.

3.0 EXPERIMENT DEMONSTRATION

3.1 DEMONSTRATION SETUP

The three basic requirements needed to set up the tri-service demonstration are hardware, software, and network communications. The following sections describe the requirements needed to support the demonstration.

3.1.1 Hardware Configuration

The tri-service experiment address the issue of heterogeneity, i.e., the integration of a variety of dissimilar resources. The intention of the experiment is to use existing resources available at the three sites. Cronus, which operates on a variety of different hardware bases and operating systems, provided the "glue" to bring the different systems at NRaD, RL, and CECOM together to demonstrate the experiment. The hardware bases used at the three sites comprised the following:

Hardware Family	Operating System
Sun Workstations (Sun 3, Sun 4, Sun 386i)	SunOS (UNIX)
Masscomp	Realtime UNIX
Symbolics	Genera
DEC VAX	VMS/Ultrix
HP	HP

3.1.1.1 Minimum Demonstration Layout

The minimum requirement needed to demonstrate the Tri-Service Distributed Technology Experiment is two machines, of which at least one has a display monitor. The monitor requirement is for the JDL Status Display Manager and User Interface, which are graphical display components. Ideally, the User Interface's map display embodies an entire display monitor; however, the map can be displayed in a GraphicsTool window (Sun Workstation) with the JDL Status Display Manager on one monitor. The other experiment components can be distributed among the two machines running in the background.

The minimum requirement of the hardware layout is feasible in a one-site or two-site demonstration but fails to truly demonstrate the full potential of the distributed capability of the experiment. Section 3.1.1.2 describes the ideal recommended layout to demonstrate the tri-service experiment.

3.1.1.2 Recommended Demonstration Layout

The ideal recommended layout to demonstrate the Tri-Service Distributed Technology Experiment would be one machine per experiment component in a three-site demonstration (figure 7). In addition, the JDL Status Display Manager and User Interface each would have their own monitor for their graphical displays. It is recommended that the JDL Status Display Manager and User Interface be displayed on color monitors; however, these two graphical components can be displayed on monochrome monitors.

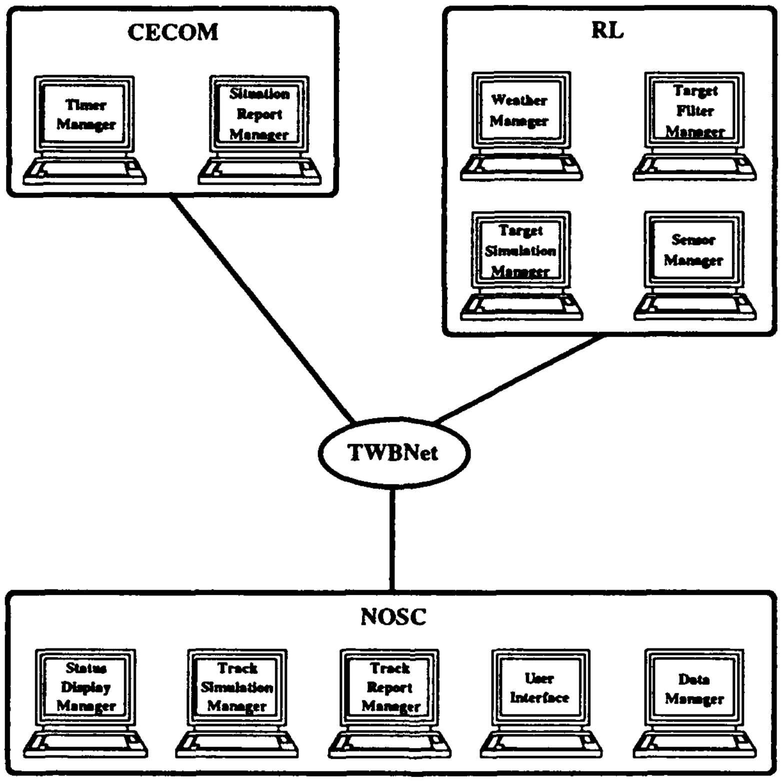


Figure 7. Sample demonstration layout.

Many other possible layouts can be used to demonstrate the Tri-Service Distributed Technology Experiment based on hardware availability and the number of participating sites. If hardware availability is limited and multiple experiment components have to be on the same machine, it is recommended that these components do not perform operations on each other. For example, it is not recommended to have the Track Simulation Manager and Track Report Manager running on the same machine in which the Track Simulation Manager invokes the **TrackUpdate** operation on the Track Report Manager. Such intra-invocation fails to demonstrate the distributed capability of the experiment. The Timer Manager and Track Report Manager can be on the same machine because they do not interact with each other (see figure 1 showing intercommunication links among the experiment components). In addition, it may be necessary to have multiple components on the same machine when demonstrating replication and duplication of components when addressing survivability capability. For example, the Track Report Manager and Target Simulation Manager have replication capability and multiple copies of these managers can be running on different machines in conjunction with other components.

3.1.2 System and Software Configuration

Cronus provides every object with a unique name when the object is created. This name is guaranteed to be unique over the entire collection of Cronus objects on all hosts. It is a structured 96-bit string, of which 80 bits make the object unique and 16 bits identify the object's type. These numbers are the Cronus **Unique Identifiers (UIDs)**. Cronus generates the 80-bit **unique numbers** (UNOs) that are part of UIDs. The Cronus configuration for the Tri-Service Distributed Technology Experiment is **UNO30**.

Every Cronus manager has to be assigned a unique type number associated with its type definition name. The following defines the type definition of the experiment managers and their abbreviation names:

260	JDLSimulation jdlobj
261	JDLTimer jdltime
262	JDLTargetSimulation jdltarg
263	JDLWeatherData jdlwxdata
264	JDLWeatherForecast jdlfcast
265	JDLSimulationData jdldata
266	JDLSensor jdlsens
267	JDLStatusDisplay jdldisp
268	JDLTargetFilter jdltfil
270	JDLSimulationDataAAObject jdldata_aa
271	JDLSimulationDBDataObject jdldata_db
320	JDLTrackSimulation jdltrack tracksim parser
321	JDLTrackReportData jdltrm trackrpt trm
322	JDLTrackTableIndex jdltrmidx trmidx
540	JDLSituation_Report jdlsit sitrep

Each JDL system at each site has a **jdldemo** account. Once the user logs in, he is in the jdl directory. Under the jdl directory are two main directories, *DEMO* and *TESTBED*. The *DEMO* directory contains subdirectories of the experiment component software that is demonstrable. The *TESTBED* directory contains subdirectories of the experiment component software that is being implemented, modified, and tested. The subdirectories structure is the same in *DEMO* and *TESTBED*:

commands: This directory contains initialization files for the Air Force components.
jdlobj: This directory contains definition code to the JDL objects.
sensor: This directory contains the source code for the Sensor Manager.
simdata: This directory contains the source code for the Simulation Data Manager.
sitrep: This directory contains the source code for the Situation Report Manager.

statdisp: This directory contains the source code for the JDL Status Display Manager.
targfil: This directory contains the source code for the Target Filter Manager.
targsim: This directory contains the source code for the Target Simulation Manager.
timer: This directory contains the source code for the Timer Manager.
trackrpt: This directory contains the source code for the Track Report Manager.
tracksim: This directory contains the source code for the Track Simulation Manager.
ui: This directory contains the source code for the User Interface.
weather: This directory contains the source code for the Weather Manager.

3.1.3 Network Connectivity

The network connectivity for the three sites is over the Defense Research Internet (DRI) via the Terrestrial Wideband Network (TWBNet) (figure 8). The DRI is a national network for research and has replaced the ARPANet. In the DRI, local-area networks (LANs) are connected directly to the backbone network. TWBNet, developed and installed by BBN, is the first phase of the DRI. It uses the cross-country T1 trunks from the DARPA National Networking Testbed (NNT) to support high-speed networking (fiber-optic technology), to provide connectivity among academic and government sites, and to support a testbed for Internet protocol development and experimentation with applications.

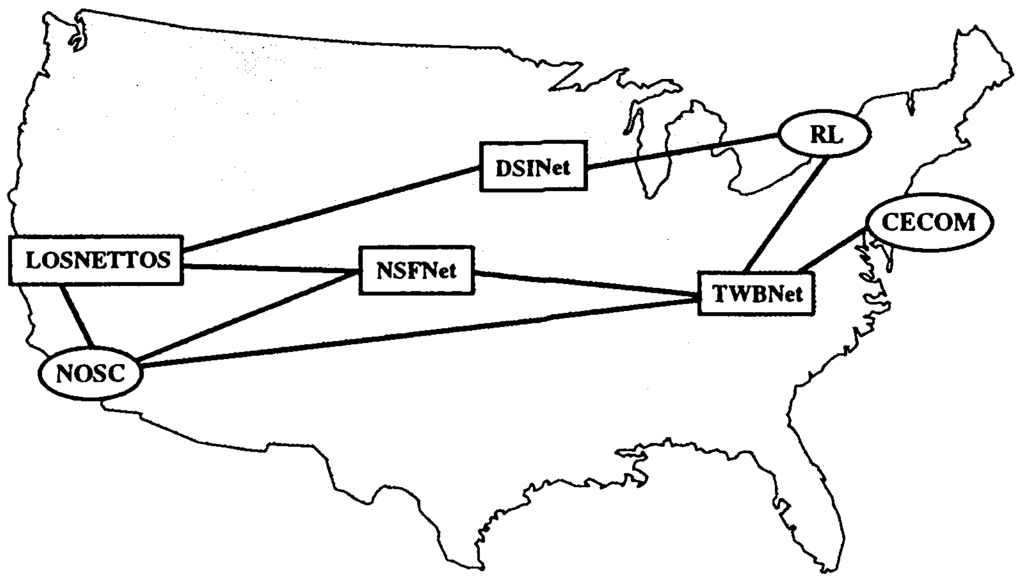


Figure 8. DRI: network connectivity.

3.2 AUTOMATIC STARTUP OF EXPERIMENT MANAGERS

The Tri-Service Distributed Technology Experiment provides an automatic startup command file, *rootmenu* (see appendix C.1). The rootmenu drives the automatic startup of the experiment managers in a window menu format for the Sun workstation. It is located in the *bin* subdirectory of the *jdl* directory.

Using the step menu, the user first selects the hardware configuration to be used in the demonstration. The configuration identifies at which physical site each of the software managers will run and the specific host machine at that site at which the manager will reside. There are five configuration files: *config.CERLNO, config.CERL, config.RLNO, config.RL,* and *config.NO* (see appendix C.2). The *config.CERLNO* file setups the hardware configuration for demonstration at all three sites (CECOM, RL, NRaD). A two-site demonstration is defined in the files, *config.CERL* (CECOM and RL) and *config.RLNO* (RL and NRaD). The *config.NO* and *config.RA* files define one-site demonstration configuration at NRaD and RL, respectively.

After setting up the hardware configuration, the next step is to start up the experiment managers. The starting of the managers are defined in the file *jdlmgrstart* with a parameter of *all, AF, ARMY,* or *NAVY* (see appendix C.3). The parameters indicate to start all the managers, only the Air Force managers, only Army manager, or only Navy managers. The *jdlmgrstart* file starts the service of each manager on the machine specified in the hardware configuration (e.g., config.CERLNO). Each manager is started by using the Cronus command *startservice.*

After the managers under the designated condition are started, the appropriate managers are linked to the Timer Manager for simulation time by calling the *jdllinktime* command file (see appendix C.4). The user has the option to link the managers to the JDL Status Display Manager, which is defined in the command file *jdllinkdisp* (see appendix C.5). Once everything is set, the experiment simulation is started in the file **jdlsimstart** (see appendix C.6).

3.3 MANUAL STARTUP OF EXPERIMENT MANAGERS

The manual startup is for users who either cannot use the automatic startup command file (e.g., window menu format for the Sun workstation is unavailable) or wishes to use only the startup portion of the experiment (e.g., for testing purposes). The user should first decide what type of demonstration will be given; that is, one-site, two-site, or three-site demonstration. If it is going to be either a two-site or three-site demonstration, then the user needs to decide how the components are going to be distributed among the sites; that is, which manager or client running at which site.

There are two options to activate the managers manually:

<u>Option A</u> - A manager can be activated via the Cronus command *startservice* if the manager is installed on the host the user wishes to run the manager. The user can execute the following Cronus command to find out where the experiment managers have been installed at the three sites:

```
% cronus /showkernel /all
```

Option B - A manager can be activated in the directory it resides in if the manager is not installed on the host the user wishes to run the manager. (Option B is described in a UNIX environment.)

3.3.1 JDL Status Display Manager

Option A

```
% cronus startservice jdldisp@host
```

@host is the hostname of the machine on which the JDL Status Display Manager is installed.

Option B

```
% cd statdisp
% jdldisp /dbcreate=0
% jdldisp &
```

To start the JDL Status Display Manager, go to the directory that contains the executable **jdldisp**. The JDL Status Display Manager is a Cronus manager; thus, the object database for its object type needs to be created. Check the directory for the file *objectdb.267*. If the file does not exist, execute */dbcreate* to create the object database. The manager can either be activated in the background (jdldisp &) or in the foreground (jdldisp /interactive). Once the manager is activated, a graphics display will appear on the monitor screen.

3.3.2 Timer Manager Startup

Option A

```
% cronus startservice jdltime@host
```

@host is the hostname of the machine on which the Timer Manager is installed.

Option B

```
% cd timer
% timer /dbcreate=0
% timer &
```

To start the Timer Manager, go to the directory that contains the executable **timer**. Check if the *objectdb.261* file exists in the directory. It is the object database for the Timer Manager. If the file does not exist, the object database for its object type needs to be created by executing */dbcreate*. The manager can either be activated in the background (timer &) or in the foreground (timer /interactive).

3.3.3 Track Report Manager Startup

Option A

```
% cronus startservice jdltrm@host
```

@host is the hostname of the machine on which the Track Report Manager is installed.

Option B

```
% cd trackrpt
% trackrpt /dbcreate=0
% trackrpt &
```

To start the Track Report Manager, go to the directory that contains the executable **trackrpt**. The Track Report Manager stores Navy data in its object database. The object database should be removed and recreated for each demonstration. The Track Report Manager manages two databases (types 321 and 322). If *objectdb.321* and *objectdb.322* exist in the directory, remove them and then execute */dbcreate* to create the new object databases. The Track Report Manager can be activated in the background (trackrpt &) or in the foreground (trackrpt /interactive). A foreground activation will print debugging and error messages from the manager onto the screen. In the background, messages from the manager will be stored in the *log.txt* file.

The Track Report Manager has replication capability. To activate other Track Report Managers on other host machines (Cronus will not allow multiple copies of a Cronus manager to run on the same host machine) is similar to activating the first Track Report Manager, except that */dbreplicate* is used instead of */dbcreate*.

```
% cd trackrpt
% trackrpt /dbreplicate=0
% trackrpt &
```

3.3.4 Track Simulation Manager Startup

Option A

```
% cronus startservice jdltrack@host
```

@host is the hostname of the machine on which the Track Simulation Manager is installed.

Option B

```
% cd tracksim
% tracksim /dbcreate=0
% tracksim /interactive
```

To activate the Track Simulation Manager, go to the directory that contains the executable **tracksim**. Check if the *objectdb.320* file exists in the directory. It is the object database for the Track Simulation Manager. If the file does not exist, the object database for its object type needs to be created by executing */dbcreate*. The Track Simulation Manager can be activated in the background (tracksim &) or in the foreground (tracksim /interactive). A foreground activation will print debugging and error messages from the manager onto the screen. In the background, messages from the manager will be stored in the *log.txt* file.

3.3.5 Target Simulation Manager Startup

<u>Option A</u>

```
% cronus startservice jdltarg@host
```

@host is the hostname of the machine on which the Target Simulation Manager is installed.

<u>Option B</u>

```
% cd targsim
% targsim /dbcreate=0
% initmedtarg
% targsim &
```

To activate the Target Simulation Manager, go to the directory that contains the executable **targsim**. Check if the *objectdb.262* file exists in the directory. It is the object database for the Target Simulation Manager. If the file does not exist, the object database for its object type needs to be created by executing */dbcreate*. If the user needed to create the manager's object database, the target data also need to be created by executing the command file *initmedtargs* (see appendix B.3). This command file creates target data and stores data objects in the Cronus directory ":jdl:targets."

If the object database already exists, the user only needs to activate the Target Simulation Manager either in the background (targsim &) or in the foreground (targsim /interactive). In the foreground, any messages from the manager will be printed on the monitor screen. In the background, any messages will be stored in the *log.txt* file.

3.3.6 Target Filter Manager Startup

<u>Option A</u>

```
% cronus startservice jdltfil@host
```

@host is the hostname of the machine on which the Target Filter Manager is installed.

<u>Option B</u>

```
% cd targfil
% targfil /dbcreate=0
% targfil &
```

To activate the Target Filter Manager, go to the directory that contains the executable **targfil**. Check if the *objectdb.268* file exists in the directory. It is the object database for the Target Filter Manager. If the file does not exist, the object database for its object type needs to be created by executing */dbcreate*. The Target Filter Manager can be activated in the background (targfil &) or in the foreground (targfil /interactive). A foreground activation will print debugging and error messages from the manager onto the screen. In the background, messages from the manager will be stored in the *log.txt* file.

3.3.7 Weather Manager Startup

<u>Option A</u>

```
% cronus startservice jdlfcast@host
```

@host is the hostname of the machine on which the Weather Manager is installed.

<u>Option B</u>

```
% cd weather
% weather /dbcreate=0
% weather &
```

To activate the Weather Manager, go to the directory that contains the executable weather. Check if the *objectdb.264* file exists in the directory. It is the object database for the Weather Manager. If the file does not exist, the object database for its object type needs to be created by executing */dbcreate*. The Weather Manager can be activated in the background (weather &) or in the foreground (weather /interactive). A foreground activation will print debugging and error messages from the manager onto the screen. In the background, messages from the manager will be stored in the *log.txt* file.

3.3.8 Sensor Manager Startup

<u>Option A</u>

```
% cronus startservice jdlsen@host
```

@host is the hostname of the machine on which the Sensor Manager is installed.

<u>Option B</u>

```
% cd sensor
% sensor /dbcreate=0
% initmedsensor
% sensor &
```

To activate the Sensor Manager, go to the directory that contains the executable **sensor**. Check if the *objectdb.266* file exists in the directory. It is the object database for the Sensor Manager. If the file does not exist, the object database for its object type needs to be created by executing */dbcreate*. If the user needs to create the manager's object database, the sensor data also need to be created by executing the command file *initmedsensor* (see appendix B.4). This command file creates sensor data and stores the data object in the Cronus directory ":jdl:sensors".

If the object database already exists, the user only needs to activate the Sensor Manager either in the background (sensor &) or in the foreground (sensor /interactive). In the foreground, any messages from the manager will be printed on the monitor screen. In the background, any messages will be stored in the *log.txt* file.

3.3.9 Situation Report Manager Startup

Option A

```
% cronus startservice jdlsit@host
```

@host is the hostname of the machine on which the Situation Report Manager is installed.

Option B

```
% cd sitrep
% sitrep /dbcreate=0
% sitrep &
```

To start the Situation Report Manager, go to the directory that contains the executable **sitrep**. The Situation Report Manager stores Army data in its object database. The object database should be removed and recreated for each demonstration. If *objectdb.540* exists in the directory, remove it and then execute */dbcreate* to create a new object database. The Situation Report Manager can be activated in the background (sitrep &) or in the foreground (sitrep /interactive). A foreground activation will print any messages from the manager onto the screen. In the background, messages from the manager will be stored in the *log.txt* file.

3.3.10 Simulation Data Manager Startup

Option A

```
% cronus startservice jdldata@host
```

@host is the hostname of the machine on which the Simulation Data Manager is installed.

Option B

```
% cd simdata
% simdata /dbcreate=0
% simdata &
```

3.3.11 Summary of Manual Startup

The following is a summary of the two options startup of the tri-service experiment manually as described above.

Option A

```
% cronus startservice jdltrm@host
% cronus startservice jdltrack@host
% cronus startservice jdlsit@host
% cronus startservice jdlsen@host
% cronus startservice jdltarg@host
% cronus startservice jdltfil@host
% cronus startservice jdlfcast@host
% cronus startservice jdltime@host
% cronus startservice jdlstatdisp@host
% cronus startservice jdldata@host
```

__Option B__

```
% cd statdisp
% jdldisp /dbcreate=0
% jdldisp &

% cd timer
% timer /dbcreate=0
% timer &

% cd trackrpt
% trackrpt /dbcreate=0
% trackrpt &

% cd tracksim
% tracksim /dbcreate=0
% tracksim &

% cd targsim
% targsim /dbcreate=0
% initmedtargs
% targsim &

% cd targfil
% targfil /dbcreate=0
% targfil &

% cd weather
% weather /dbcreate=0
% weather &

% cd sensor
% sensor /dbcreate=0
% initmedsensor
% sensor &

% cd sitrep
% sitrep /dbcreate=0
% sitrep &

% cd simdata
% simdata /dbcreate=0
% simdata &
```

3.3.12 Experiment Simulation Startup

After the managers have been activated, go into *tropic* to set up the experiment simulation.

```
% cronus tropic
command: on {jdltrm}@host InitializeDB
command: on {jdltime}@host create
command: set timer on $obj
command: on {jdltrack}@host LinkToTimer $timer
command: on {jdlsit}@host LinkToTimer $timer
command: on {jdltarg}@host LinkToTimer $timer
command: on {jdlsens}@host LinkToTimer $timer
command: on {jdlfcast}@host LinkToTimer $timer
command: on {jdltrm}@host LinkToDisplay
command: on {jdltime}@host LinkToDisplay
command: on {jdltrack}@host LinkToDisplay
command: on {jdlsit}@host LinkToDisplay
command: on {jdltarg}@host LinkToDisplay
command: on {jdlsens}@host LinkToDisplay
command: on {jdlfcast}@host LinkToDisplay
command: on {jdltfil}@host LinkToDisplay
command: on {jdldata}@host LinkToDisplay
command: on ":jdl:sensors:nosc_scooby.MedSea" Operatesensor 0
command: on {jdlsit}@host ReadSitRep armyfile
command: on {jdltrack}@host ReadTracks navyfile
command: on $timer start
```

3.4 USER INTERFACE ACTIVATION

The User Interface is a Cronus client that invokes the Simulation Data Manager for tri-service data for display. Because it is not a Cronus manager, the User Interface can only be activated in the directory it resides in.

To activate the User Interface, go to the directory that contains the executable **graphics** (*ui* directory) and activate the executable.

```
% cd ui
% graphics
```

Successful activation of the User Interface will display the name of the Tri-Service Distributed Technology Experiment.

4.0 REFERENCES

Berets, J. C., and R. M. Sands, *Introduction to Cronus*, Report No. 6986, BBN Systems and Technologies Corporation, January 1989.

BBN Systems and Technologies Corporation, *RELEASE NOTICE Cronus Release 1.5*, 1 May 1990.

BBN Systems and Technologies Corporation (The Diamond Group), *Diamond Multimedia Toolkit*, 16 February 1989.

Gadbois, M. J., *Distributed Operating System Experiment (DOSE) Application User's Manual*, Technical Document 1801, Naval Ocean Systems Center, February 1990.

Gadbois, M. J., and M. F. Barrett, *JDL Tri-Service User Interface Source Code*, Technical Document 1733, Naval Ocean Systems Center, January 1990.

Gadbois, M. J., and A. M. Newton, "Tri-Service Distributed Technology Experiment," *1990 Symposium on Command and Control Research Proceeding*, 12–14 June 1990.

GOTS 1.0.5 Programming Guide (Alternate Environment), Space and Naval Warfare Systems Command (SPAWAR), PMW 162 Naval Tactical Command System (NTCS), 21 January 1991.

Newton, A. M., *Installation and Operational Instructions for the C2 Internet Application*, Rome Laboratory, Internal White Paper, 23 August 1989.

APPENDIX A

APPENDIX A.1: TRACK REPORT MANAGER DATA OBJECT REPRESENTATION

File Name: trkrpt.def

```
cantype TRACKOBJECT
        representation is TrackObject:
        record
                track_rpt:          TRACKREPORT
                        annote "track report information";
                track_rpt_plus:  TRACKREPORTPLUS
                        annote "additional track report information";
        end TRACKOBJECT
        annote "Each Track Report Manager object contains the full
                information about a track";


cantype TRACKREPORT
        representation is TrackReport:
        record
                trk_num:          U16I
                        annote "track range of track number in octal
                                        0000          Illegal number
                                        0001-0076 PU address
                                        0077          Illegal number
                                        0100-0176 RU address
                                        0177          PU/RU collect.addr
                                        0200-7776 track numbers
                                        7777          internal - illegal";
                lat_track:        F32
                        annote "latitude of track (minutes),
                                range of latitude
                                        -5400 to 5400 minutes";
                long_track:       F32
                        annote "longitude of track (minutes),
                                range of longitude
                                        -10800 to 10800 minutes";
                bearing:          F32
                        annote "bearing from dlrp (degrees),
                                values of bearing
                                        0 to 360 degress";
                lat_dlrp:         F32
                        annote "latitude of dlrp (minutes),
                                range of latitude
                                        -5400 to 5400 minutes";
                long_dlrp:        F32
                        annote "longitude of dlrp (minutes),
                                range of longitude
                                        -10800 to 10800 minutes";
                dep_high:         F32
                        annote "depth or height (feet),
                                range of depth/height
                                        0 to ~127,000 feet";
                platform_type:  ASC
                        annote "type of platform,
                                values of data type
                                        A = Air
                                        L = Land
                                        S = Surface
                                        U = Subsurface
                                        x = Unreported";
                cat:                 ASC
                            annote "identity or catagory,
                                    values of catagory
                                            F = Friendly
                                            H = Hostile
                                             I = Interceptor
                                            N = Nonsubmarine
                                            S = Special
                                            U = Unknown
                                             x = Unreported";
        end TRACKREPORT;


    cantype TRACKREPORTPLUS
        representation is TrackReportPlus:
        record
                time:              S32I
                          annote "greenwich mean time";
                trkqa:             U16I
                          annote "track quality,
                                  values of track quality
                                          0 to 7";
                lat_tdir:          ASC
                          annote "direction north or south,
                                  values of latitude direction
                                          N = North, S = South";
                long_tdir:          ASC
                          annote "direction east or west,
                                  values of longitude direction
                                          E = East, W = West";
                course:            F32
                          annote "course (degrees),
                                  values of course
                                          0 to 360 degrees";
                speed:             F32
                          annote "speed (dm/hr),
                                  range of speed
                                          0 to ~3571-7/8 dm/hr";
                range:             F32
                          annote "range from dlrp (naut. miles)";
                nuclear:           ASC
                          annote "nuclear,non-nuclear,unknown,
                                  values of nuclear
                                          Y = Yes, nuclear
                                          N = No, not nuclear
                                          U = Unknown
                                           x = Unreported";
        end TRACKREPORTPLUS;
```

APPENDIX A.2: SENSOR MANAGER DATA OBJECT REPRESENTATION

File Name: sensor.typedef

```
cantype SENSORVARIABLE
          representation is SENSORvariable:
          record
                    Name:              ASC;
                    Type:              SENSORTYPE;
                    xyResolution:      S16I;
                    FrmDefs:           array of FRMDEF;
          end SENSORvariable
          annote "The sensor is fully described by the SENSORvariable structure. It
                    contains: the label for the sensor (Name), the characteristic of the
                    sensor (Type), the resolution parameter for the sensor (xyResolution)
                    in deciseconds, and the list of sensor schedules (FrmDefs).";


cantype SENSORTYPE
          representation is SENSORtype:
                    { radar, infrared, sigint }
          annote "SENSORtype is an enumerated type that designates the characteristics
                    of the sensor.";


cantype FRMDEF
          representation is FRMdef:
          record
                    tStart:            EINTERVAL;
                    tEnd:              EINTERVAL;
                    tInc:              EINTERVAL;
                    Location:          LOCATION3D;                              (See Appendix A.9)
                    Region:            EREGION;                                 (See Appendix A.9)
          end FRMdef
          annote "A frame is the fundamental unit of observation used by the sensor.
                    he FRMdef structure defines the frame unit. Each frame is defined
                    by: the starting time (tStart), the ending time (tEnd), the interval
                    of time between observations (tInc), the location of the sensor
                    (Location), and the region of space that is being observed (Region).";
```

APPENDIX A.3: TARGET SIMULATION MANAGER DATA OBJECT REPRESENTATION

File Name: targsim.typedef

```
cantype TARGETVARIABLE
        representation is TARGETvariable:
        record
                Name:           ASC;
                TargType:       TARGETTYPE;
                TargID:         TARGETIDENT;
                TargVal:        ETHREAT;
                ControlNum:     U16I;
                FirstVisible:   EBOOL;
                Paths:          array of ESEGMENT;                    (See Appendix A.9)
        end TargetVariable
        annote "A target is comprised of: (1) a name, (2) a type, (3) an identity
                keyword, (4) a threat rating, (5) a control number (this is a PATCH
                used to make these objects compatible with the NAVY UserInterface
                client. The NAVY interface should be changed to identify and log
                objects based on the UID not a control number), and (6) a list of
                line segments. Each segment is considered to be the path that the
                target travelled during a particular time interval.";


cantype TARGETTYPE
        representation is TARGETtype:
                { Jeep, Tank, Airplane, Building, Airfield, Truck, Bridge,
                  PowerPlant, Railroad, Dam, Ship, Missile }
        annote "An enumerated type to describe the type of target being represented.";


cantype TARGETIDENT
        representation is TARGETident:
                { AirTARGET, MobileGroundTARGET,
                  StationaryGroundTARGET, BaseTARGET, UnknownTARGET }
        annote "An enumerated type to describe the target identity.";


cantype ETHREAT
        representation is THREAT:
                { Friendly, Hostile, Unknown }
        annote "An enumerated type to describe the intentions of the target.";
```

APPENDIX A.4: TARGET FILTER MANAGER DATA OBJECT REPRESENTATION

File Name: targfil.typedef

```
cantype FILTERREPORT
        representation is FILTERreport:
        record
                TotalDetections :       U16I;
                RealDetections :        U16I;
                SemiRealDetections :  U16I;
                FakeDetections :        U16I;
                OpsProcessed :          U16I;
                SensorStats :             array of SENSORDATA;
        end FILTERreport
        annote "This cantype record holds all of the statistical information
                for the Target Filter ReportStatus - the total numb_ _f
                detections processed, % of real, semi-real, and false
                detections, the total number of operations processed, and
                who sent those operations along with a count of the
                number of operations sent.";


cantype SENSORDATA
        representation is SENSORdata:
        record
                SensorID :               ASC;
                NumOps  :                U16I;
        end SENSORdata
        annote "This cantype record holds Sensor IDs and the number of
                operations sent by the respective Sensors.";
```

APPENDIX A.5: WEATHER MANAGER DATA OBJECT REPRESENTATION

File Name: weather.typedef

type *JDLWeatherForecastObject*

```
cantype WXCASTDATA
        representation is WXCASTdata:
        record
                ForecastDate:         EINTERVAL;
                BeginValidPeriod:     EINTERVAL;
                EndValidPeriod:       EINTERVAL;
                ForecastData:         WXINFO;
        end WXInfo
        annote "A weather forecast is comprised of: (1) weather information for a
                map location, (2) a date when that forecast was made, and (3) dates
                that define the utility of the information.";
```

type *JDLWeatherDataObject*

```
cantype WXINFO
        representation is WXinfo:
        record
                Location:                 LOCATION2D;                    (See Appendix A.9)
                Ceiling:                  U16I;
                CloudCoverage:            CLOUDCOVERAGE;
                CloudTypes:               array of CLOUDTYPE;
                SurfaceVisibility:        U32I;
                VisibilityRestrictions:   ASC;
                Weather:                  WEATHERTYPE;
                BarometricPressure:       U16I;
                Temperature:              S16I;
                DewPoint:                 S16I;
                SurfaceWindDirection:     U16I;
                AvgSurfaceWindSpeed:      U16I;
                MaxSurfaceWindSpeed:      U16I;
                Remarks:                  ASC;
                Station:                  ASC;
        end WXinfo
        annote "The weather information contained in a concise package. This can
                become the basis for either a weather report or a weather forecast.";


cantype CLOUDCOVERAGE
        representation is CLOUDcov:
                { zero, light, heavy }
        annote "An enumerated type to describe the amount of cloud coverage.";


cantype CLOUDTYPE
        representation is CLOUDtype:
                { cirrus, nimbus, cumulus, stratus,
                  cirrocumulus, stratocumulus, altocumulus,
                  cirrostratus, nimbostratus, altostratus,
                  cumulonimbus }
        annote "An enumerated type to describe the type of cloud coverage.";


cantype WEATHERTYPE
        representation is WEATHERtype:
                { clear, clouds, fog, rain, snow }
        annote "An enumerated type to describe the general weather condition.";
```

APPENDIX A.6: TIMER MANAGER DATA OBJECT REPRESENTATION

File Name: timer.typedef

```
cantype TIMERSTATUS
        representation is TIMERstatus:
        record
                HostTime:           EDATE;
                GlobalTime:         EDATE;
                Adjustment:         EINTERVAL;
                AdjAvg:             S32I;
                AdjDev:             S32I;
                AdjSamples:         S32I;
                SyncInterval:       U16I;
                TimeUntilSync:      U16I;
        end TIMERstatus
        annote "Status information that is normally maintained by the manager and
                reported upon request.";


cantype TIMEROBJECT
        representation is TIMERobject:
        record
                TimerTimeRef:       EDATE;
                GlobalTimeRef:      EDATE;
                InitialValue:       EDATE;
                StopTime:           EDATE;
                StepIncrement:      EINTERVAL;
                Running:            EBOOL;
                AwaitingStop:       EBOOL;
                RateMultiplier:     U16I;
                RateDivisor:        U16I;
        end TIMERobject
        annote "The simulation clock database, with all information needed to
                produce a simulation time. Each object represents a different
                simulation clock.";
```

APPENDIX A.7: SITUATION REPORT MANAGER DATA OBJECT REPRESENTATION

File Name: sitrep.typedef

```
cantype SIT_REP
        representation is Situation_Report_Data:
        record
                Classification:             UNIT_CLASS;
                Unit_Name:                  ASC;
                Unit_Number:                U32I
                        annote "Manager-generated number (needed for user interface).";
                Unit_Type:                  UNIT_TYPE;
                Unit_Latitude:              F32;
                Unit_Longitude:             F32;
                Mission:                    UNIT_MISSION;
                Subord_comm:                EBOOL
                        annote "Implemented as a boolean, it is really a YES/NO answer.";
                Enemy_Contact:              UNIT_CONTACT;
                Weapon_Stat_Crews:          U16I
                        annote "Percentage of crew available * 100.";
                Weapon_Stat_Equipment:  U16I
                        annote "Percentage of equipment availabel * 100.";
                Eval_Status:                UNIT_STATUS;
                Epoch:                      EINTERVAL;
                DM_Uid:                     EUID
                        annote "UID of Data Manager object containing this report.";
        end SIT_REP
        annote "Contains the information necessary for a situation report for a given unit.";


cantype UNIT_CLASS
        representation is UnitClassificationEnum:
                {UNKNOWN, HOSTILE, FRIENDLY}
        annote "describes the possible classifications (friendly, hostile, or unknown)
                of a unit.";


cantype UNIT_TYPE
        representation is UnitTypeEnum:
                {MECH_INFANTRY, ARTILLERY, CAVALRY}
        annote "delineates the types of units reported on by this manager.
                For purposes of the JDL experiment, the complete list has been reduced
                to these three values.";


cantype UNIT_MISSION
        representation is UnitMissionEnum:
                {OFFENSE, DEFENSE, DIR_SUPPORT, GEN_SUPPORT}
        annote "possible missions a unit can perform.";


cantype UNIT_CONTACT
        representation is UnitEnemyContactEnum:
                {NONE, LIGHT, HEAVY}
        annote "possible levels of enemy contact.";


cantype UNIT_STATUS
        representation is UnitStatusEnum:
                {BLACK, RED, YELLOW, GREEN}
        annote "possible commander's evaluation of unit status.";
```

```
cantype SR_STATUS
          representation is Situation_Report_Status:
          record
                   Up_Since:              EDATE
                             annote "Date/Time manager was started.";
                   Linked_To_Timer:  EBOOL
                             annote "Is the manager linked to a simulation timer.";
                   Timer:                 EUID
                             annote "Associated timer object for this manager.";
          Reading_SitReps:               EBOOL
                             annote "Is the manager present reading Situation Reports from a data file.";
          end SR_STATUS
          annote "Status information returned by ReportStatus() operation.";
```

APPENDIX A.8: SIMULATION DATA MANAGER DATA OBJECT REPRESENTATION

File Name: simdata.typedef

```
cantype SIMDATA
        representation is SIMdata:
        record
                Position:        LOCATION3D;                              (See Appendix A.9)
                DataType:        DATATYPE;                                (See Appendix A.9)
                Source:          DATASOURCE;                              (See Appendix A.9)
                ThreatID:        ASC;
                Category:        ASC;
                Contents:        ESTORAGE;                                (See Appendix A.9)
        end SIMdata
        annote "The SIMdata structure will form the basic data object for the JDL
                simulation. The objective is to provide an interface for quickly
                looking up data that the UserInterface needs, while maintaining the
                object that has been entrusted to the DataManager. For quick
                retrieval of information, the Position, DataType, and Source entries
                are designed to allow the UserInterface to frame data search
                requests. These options should map directly to the options displayed
                on the UserInterface screen. The Contents entry allows the manager
                that stores data to store an arbitrary set of data (e.g. an object
                or a simple set of tracks) for use by the simulation or by the
                UserInterface.";


cantype SIMREF
        representation is SIMref:
        record
                ObjUID:          EUID;
                Position:        LOCATION3D;                              (See Appendix A.9)
                DataType:        DATATYPE;                                (See Appendix A.9)
                Source:          DATASOURCE;                              (See Appendix A.9)
                ThreatID:        ASC;
                Category:        ASC;
        end SIMref
        annote "The SIMref structure is used bye calling processes to access the
                essential elements of the RetrieveObjects search operation. More
                detailed information can be obtained about individual simdata
                objects by using the RetrieveData operation.";


cantype DATABASES
        representation is DATAbases:
                { INFORMIX, ORACLE, SYBASE, NODB }
        annote "Identify the database that will be used by the SimulationData
                manager.";
```

APPENDIX A.9: JDL OBJECT REPRESENTATION

File Name: jdlobj.typedef

```
cantype LOCATION2D = 26001
        representation is LOC2D:
        record
                Latitude:       S32I;
                Longitude:      S32I;
        end Loc
        annote "Two dimensional point in the simulation.  Longitude and Latitude are
                in deciseconds.";


cantype LOCATION3D = 26002
        representation is LOC3D:
        record
                Latitude:       S32I;
                Longitude:      S32I;
                Altitude:       S32I;
        end Loc3D
        annote "Three dimensional point in the simulation.  Longitude and Latitude are
                in deciseconds.  Altitude is in feet.";


cantype EREGION = 26003
        representation is REGION:
        record
                LowerLeft:      LOCATION3D;
                UpperRight:     LOCATION3D;
        end EREGION
        annote "A three dimensional cubic region in the simulation.";


cantype ESEGMENT = 26004
        representation is SEGMENT:
        record
                StartTime:      EINTERVAL;
                StartLoc:       LOCATION3D;
                EndTime:        EINTERVAL;
                EndLoc:         LOCATION3D;
        end ESEGMENT
        annote "A straight line representation of a target path.";


cantype TARGETDATA = 26005
        representation is TARGETdata:
        record
                TargetUID:          EUID;
                TargetLocation:     LOCATION3D;
                TargetDescription:  ASC;
                TargetType:         ASC;
                TargetNum:          U16I;
                ThreatValue:        ASC;
        end TARGETdata
        annote "This represents a generic format for simulation targets.  A target
                is defined to be any object that is detected by a sensory object
                in the simulation.  Most of the fields are ASCII to allow users to
                customize the information contained within.";
```

```
cantype DETECTIONDATA = 26006
        representation is DETECTIONdata:
        record
                OriginUID:          EUID;
                OriginName:         ASC;
                Time:                 EINTERVAL;
                snr:                  U16I;
                Target:              TARGETDATA;
        end DETECTIONdata
        annote "This represents raw data that has been seen (detected) within the
                simulation. It is the common unit of data passed by all detection
                devices (such as sensors). Detections are point based, but, each
                point provides enough information to reassemble scattered data.";


cantype DATATYPE = 26007
        representation is DATAtype:
                { WeatherData, TrackData, TargetData, SituationReport }
        annote "DATAtype is an enumerated type that describes the type of information
                passed through the simulation. It ultimately should correspond to
                the type of information viewable at the user interface. Data
                sent to the SimulationData manager MUST be identified with an
                approriate DATAtype.";
cantype DATASOURCE = 26008
        representation is DATAsource:
                { AirForce, Army, Navy }
        annote "DATAsource is an enumerated type that describes the service sites
                of the JDL simulation.";


cantype ESTORAGE = 26009
        representation is STORAGE:
        record
                ctype:          U16I;
                clength:        S32I;
                cdata:          OVEC;
        end STORAGE
        annote "The STORAGE structure holds the contents of ONE and ONLY ONE
                canonical type. Whether this is used to transport an object
                or store relevant data, the essential information for conversion
                back into the internal format is present within the structure.";


cantype CONNECTSTAT = 26010
        representation is CONNECTstat:
        record
                ConnAttempts:       S32I;
                ConnTimeouts:       S32I;
                ConnErrors:         S32I;
        end CONNECTstat
        annote "Provides debugging and statistical information on the simulation
                when used with some library tools. Used primarily by the sensor
                manager, but, could be used by any manager wishing to keep track of
                its ability to talk to other managers.";
```

APPENDIX B

APPENDIX B.1: SAMPLE *navyfile* DATA FILE

```
1
  1 2490.00    0.00  241.36       0.00 2700.00 9300.00 S F   085 0 N E   320     20.59  41.32  x
1
  1 2480.00   30.00  264.12       0.00 2700.00 9300.00 S F   145 6 N E   319     19.93  56.09  x
1
  1 2490.00   60.00  280.00       0.00 2700.00 9300.00 S F   205 6 N E   319     19.93  20.86  x
2
  5 2575.00    0.00  267.31       0.00 2700.00 9300.00 S F   210 6 N E   319     19.93  33.82  x
  1 2500.00   90.00  286.42       0.00 2700.00 9300.00 S F   265 6 N E   319     19.93  42.48  x
2
  5 2585.00   30.00  267.31       0.00 2700.00 9300.00 S F   270 6 N E   319     19.93  33.82  x
  1 2510.00  120.00  233.45       0.00 2700.00 9300.00 S F   325 6 N E   319     19.93  36.31  x
2
  5 2600.00   60.00  267.31       0.00 2700.00 9300.00 S F   330 6 N E   319     19.93  33.82  x
  1 2520.00  140.00  267.31       0.00 2700.00 9300.00 S F   385 6 N E   319     19.93  33.82  x
3
  5 2620.00   85.00  267.31       0.00 2700.00 9300.00 S F   390 6 N E   319     19.93  33.82  x
  6 2260.00 1020.00  248.68    1000.00 2700.00 9300.00 U U   440 6 N E   319     19.93  54.02  x
  1 2530.00  160.00  267.31       0.00 2700.00 9300.00 S F   445 6 N E   319     19.93  33.82  x
1
  5 2630.00  110.00  267.31       0.00 2700.00 9300.00 S F   450 6 N E   319     19.93  33.82  x
3
  6 2270.00 1010.00  248.68    1000.00 2700.00 9300.00 U U   500 6 N E   319     19.93  54.02  x
  1 2540.00  190.00  267.31       0.00 2700.00 9300.00 S F   505 6 N E   319     19.93  33.82  x
  5 2640.00  135.00  267.31       0.00 2700.00 9300.00 S F   510 6 N E   319     19.93  33.82  x
2
  1 2550.00  220.00  267.31       0.00 2700.00 9300.00 S F   565 6 N E   319     19.93  33.82  x
  5 2645.00  165.00  267.31       0.00 2700.00 9300.00 S F   570 6 N E   319     19.93  33.82  x
2
  1 2560.00  250.00  267.31       0.00 2700.00 9300.00 S F   625 6 N E   319     19.93  33.82  x
  5 2645.00  195.00  267.31       0.00 2700.00 9300.00 S F   630 6 N E   319     19.93  33.82  x
2
  1 2570.00  280.00  267.31       0.00 2700.00 9300.00 S F   685 6 N E   319     19.93  33.82  x
  5 2645.00  225.00  267.31       0.00 2700.00 9300.00 S F   690 6 N E   319     19.93  33.82  x
2
  5 2645.00  255.00  267.31       0.00 2700.00 9300.00 S F   750 6 N E   319     19.93  33.82  x
  2 2570.00  300.00  248.68     700.00 2700.00 9300.00 A F   755 6 N E   319     19.93  54.02  x
3
  6 2280.00 1000.00  248.68    1000.00 2700.00 9300.00 U U   790 6 N E   319     19.93  54.02  x
  2 2570.00  330.00  248.68    1000.00 2700.00 9300.00 A F   805 6 N E   319     19.93  54.02  x
  5 2645.00  285.00  267.31       0.00 2700.00 9300.00 S F   810 6 N E   319     19.93  33.82  x
1
  2 2570.00  360.00  248.68    3000.00 2700.00 9300.00 A F   855 6 N E   319     19.93  54.02  x

2
   2 2565.00  390.00  248.68    6000.00 2700.00 9300.00 A F   905 6 N E   319     19.93  54.02  x
  10 2645.00  305.00  248.68     900.00 2700.00 9300.00 A F   915 6 N E   319     19.93  54.02  x
2
   2 2565.00  410.00  248.68   10000.00 2700.00 9300.00 A F   955 6 N E   319     19.93  54.02  x
  10 2645.00  330.00  248.68    1500.00 2700.00 9300.00 A F   965 6 N E   319     19.93  54.02  x
3
   6 2290.00  990.00  248.68    1000.00 2700.00 9300.00 U H   990 6 N E   319     19.93  54.02  x
   2 2555.00  440.00  248.68   10000.00 2700.00 9300.00 A F  1005 6 N E   319     19.93  54.02  x
  10 2645.00  360.00  248.68    2500.00 2700.00 9300.00 A F  1015 6 N E   319     19.93  54.02  x
3
   2 2550.00  470.00  248.68   10000.00 2700.00 9300.00 A F  1055 6 N E   319     19.93  54.02  x
  10 2640.00  390.00  248.68    5000.00 2700.00 9300.00 A F  1065 6 N E   319     19.93  54.02  x
   6 2310.00  980.00  248.68    1000.00 2700.00 9300.00 U H  1070 6 N E   319     19.93  54.02  x
3
   2 2540.00  490.00  248.68   10000.00 2700.00 9300.00 A F  1105 6 N E   319     19.93  54.02  x
  10 2635.00  420.00  248.68    8000.00 2700.00 9300.00 A F  1115 6 N E   319    .19.93  54.02  x
   6 2320.00  960.00  248.68    1000.00 2700.00 9300.00 U H  1120 6 N E   319     19.93  54.02  x
2
   2 2535.00  510.00  248.68   10000.00 2700.00 9300.00 A F  1155 6 N E   319     19.93  54.02  x
  10 2630.00  450.00  248.68    8000.00 2700.00 9300.00 A F  1165 6 N E   319     19.93  54.02  x
2
   2 2530.00  530.00  248.68   10000.00 2700.00 9300.00 A F  1205 6 N E   319     19.93  54.02  x
  10 2620.00  480.00  248.68    8000.00 2700.00 9300.00 A F  1215 6 N E   319     19.93  54.02  x
1
   2 2520.00  550.00  248.68   10000.00 2700.00 9300.00 A F  1255 6 N E   319     19.93  54.02  x
1
   2 2510.00  570.00  248.68   10000.00 2700.00 9300.00 A F  1305 6 N E   319     19.93  54.02  x
```

APPENDIX B.2: SAMPLE *armyfile* DATA FILE

```
00:06:20.000        Infantry_#1         2800      760HIOYNY 100       100
00:07:30.000        Infantry_#1         2795      765HIOYNY 100       100
00:09:25.000        Infantry_#1         2790      770HIOYNY 100       100
00:11:25.000        Infantry_#1         2785      775HIOYNY 100       100
00:12:35.000        Infantry_#1         2780      780HIOYNY 100       100
00:13:25.000        Infantry_#2         2695      780FIDYNG 100       100
00:14:10.000        Infantry_#1         2775      780HIOYNY 100       100
00:15:10.000        Infantry_#2         2700      780FIDYNG 100       100
00:16:00.000        Cavalry_#1          2690      940HCOYNY 100       100
00:16:05.000        Infantry_#1         2770      780HIOYNY 100       100
00:16:50.000        Infantry_#2         2705      780FIDYNG 100       100
00:17:00.000        Cavalry_#2          2690      840FCSYNG 100       100
00:17:40.000        Infantry_#1         2765      780HIOYNY 100       100
00:17:50.000        Cavalry_#1          2685      940HCOYNY 100       100
00:18:30.000        Infantry_#2         2710      780FIDYNG 100       100
00:18:40.000        Infantry_#1         2760      780HIOYNY 100       100
00:18:40.000        Cavalry_#1          2680      935HCOYNY 100       100
00:19:45.000        Infantry_#2         2715      780FIDYNG 100       100
00:20:05.000        Cavalry_#1          2680      935HCOYNY 100       100
00:20:10.000        Infantry_#1         2755      780HIOYLR 100       100
00:20:25.000        Cavalry_#2          2685      850FCSYNG 100       100
00:21:00.000        Infantry_#2         2720      780FIDYLR 100       100
00:21:50.000        Infantry_#1         2750      780HIOYLR 100       100
00:22:00.000        Cavalry_#1          2675      935HCOYNY 100       100
00:22:05.000        Cavalry_#2          2680      855FCSYNG 100       100
```

APPENDIX B.3: SAMPLE *initmedtargs* COMMAND DATA FILE

```
# $Revision$
#! /bin/csh
set noglob

echo " "
echo "SAVE CURRENT CRONUSDIRECTORY - SET TO TARGETSIMULATION DIRECTORY"
set DIR=`cronus getwdir`
cronus setwdir :jdl:targets

echo " "
echo "SET ACL OF GENERIC OBJECT"
cronus tropic << xxxEOFxxx
on {group} LookupGroup JDLSimulation
on {jdltarg} addtoacl ((\$GroupUID all))
quit
xxxEOFxxx

echo " "
echo "CREATING TARGET"

set TYPE=Ship
set NAME=EnemyShip1
set CLASS=Hostile
set LAT=1332000
set LONG=612000
set ALT=0

cronus tropic << xxxEOFxxx
newtype jdltarg
on {jdltarg} create $NAME $TYPE $CLASS "00:00:30" ($LAT, $LONG, $ALT) /Start
on :jdl:targets CreateEntry \$Object /EntryName=$NAME
xxxEOFxxx

echo Catalogued target is:  :jdl:targets:$NAME
echo Set AccessControl and Add Segments for that target.
```

```
cronus tropic << xxxEOFxxx
on {group} LookupGroup JDLSimulation
on $NAME addtoacl ((\$GroupUID all))
on $NAME moveto "00:01:20" (1344000, 600000, 0)
on $NAME moveto "00:02:20" (1356000, 588000, 0)
on $NAME moveto "00:03:20" (1368000, 576000, 0)
on $NAME moveto "00:04:20" (1380000, 564000, 0)
on $NAME moveto "00:05:20" (1383000, 552000, 0)
on $NAME moveto "00:06:20" (1386000, 540000, 0)
on $NAME moveto "00:07:20" (1392000, 534000, 0)
on $NAME moveto "00:11:40" (1392000, 522000, 0)
on $NAME moveto "00:13:50" (1404000, 510000, 0)
xxxEOFxxx

echo " "
echo "CREATING TARGET"
set TYPE=Airplane
set NAME=EnemyPlane1
set CLASS=Hostile
set LAT=1392000
set LONG=534000
set ALT=0
cronus tropic << xxxEOFxxx
newtype jdltarg
on {jdltarg} create $NAME $TYPE $CLASS "00:07:30" ($LAT, $LONG, $ALT) /noStart
on :jdl:targets CreateEntry \$Object /EntryName=$NAME
xxxEOFxxx

echo Catalogued target is:  :jdl:targets:$NAME
echo Set AccessControl and Add Segments for that target.

cronus tropic << xxxEOFxxx
on {group} LookupGroup JDLSimulation
on $NAME addtoacl ((\$GroupUID all))
on $NAME moveto "00:08:20" (1404000, 540000, 500)
on $NAME moveto "00:09:20" (1416000, 540000, 1000)
on $NAME moveto "00:10:10" (1434000, 546000, 2000)
on $NAME moveto "00:11:00" (1446000, 558000, 3000)
on $NAME moveto "00:11:50" (1464000, 576000, 6000)
on $NAME moveto "00:12:40" (1488000, 594000, 8000)
on $NAME moveto "00:13:30" (1512000, 612000, 10000)
on $NAME moveto "00:14:20" (1536000, 612000, 10000)
on $NAME moveto "00:15:10" (1560000, 600000, 10000)
on $NAME moveto "00:16:00" (1572000, 576000, 10000)
on $NAME moveto "00:16:50" (1566000, 564000, 10000)
on $NAME moveto "00:17:40" (1554000, 552000, 10000)
on $NAME moveto "00:18:30" (1542000, 528000, 10000)
on $NAME moveto "00:19:20" (1527000, 522000, 10000)
on $NAME moveto "00:20:10" (1500000, 516000, 10000)
on $NAME moveto "00:21:00" (1476000, 516000, 10000)
xxxEOFxxx

echo " "
echo "CREATING TARGET"
set TYPE=Airplane
set NAME=EnemyPlane2
set CLASS=Hostile
set LAT=1392000
set LONG=534000
set ALT=0
cronus tropic << xxxEOFxxx
newtype jdltarg
on {jdltarg} create $NAME $TYPE $CLASS "00:10:20" ($LAT, $LONG, $ALT) /noStart
on :jdl:targets CreateEntry \$Object /EntryName=$NAME
xxxEOFxxx
```

```
echo Catalogued target is:  :jdl:targets:$NAME
echo Set AccessControl and Add Segments for that target.

cronus tropic << xxxEOFxxx
on {group} LookupGroup JDLSimulation
on $NAME addtoacl ((\$GroupUID all))
on $NAME moveto "00:11:10" (1398000, 516000, 700)
on $NAME moveto "00:12:00" (1410000, 498000, 1100)
on $NAME moveto "00:12:50" (1422000, 492000, 1700)
on $NAME moveto "00:13:40" (1440000, 474000, 3000)
on $NAME moveto "00:14:30" (1452000, 456000, 8000)
on $NAME moveto "00:15:20" (1464000, 438000, 10000)
on $NAME moveto "00:16:10" (1476000, 420000, 12000)
on $NAME moveto "00:17:00" (1488000, 408000, 14000)
on $NAME moveto "00:17:50" (1500000, 396000, 16000)
on $NAME moveto "00:18:40" (1503000, 408000, 16000)
on $NAME moveto "00:19:30" (1506000, 426000, 16000)
on $NAME moveto "00:20:20" (1500000, 444000, 16000)
on $NAME moveto "00:21:10" (1488000, 456000, 16000)
on $NAME moveto "00:22:00" (1476000, 468000, 14000)
xxxEOFxxx

echo " "
echo "CREATING TARGET"
set TYPE=Airplane
set NAME=Base1Plane
set CLASS=Friendly
set LAT=1707000
set LONG=312000
set ALT=0
cronus tropic << xxxEOFxxx
newtype jdltarg
on {jdltarg} create $NAME $TYPE $CLASS "00:13:30" ($LAT, $LONG, $ALT) /noStart
on :jdl:targets CreateEntry \$Object /EntryName=$NAME
xxxEOFxxx

echo Catalogued target is:  :jdl:targets:$NAME
echo Set AccessControl and Add Segments for that target.

cronus tropic << xxxEOFxxx
on {group} LookupGroup JDLSimulation
on $NAME addtoacl ((\$GroupUID all))
on $NAME moveto "00:13:55" (1698000, 324000, 800)
on $NAME moveto "00:14:45" (1686000, 336000, 1500)
on $NAME moveto "00:15:30" (1674000, 348000, 5000)
on $NAME moveto "00:16:30" (1662000, 360000, 9000)
on $NAME moveto "00:17:00" (1650000, 372000, 12000)
on $NAME moveto "00:18:00" (1638000, 384000, 14000)
on $NAME moveto "00:19:00" (1626000, 396000, 20000)
on $NAME moveto "00:19:50" (1614000, 408000, 22000)
on $NAME moveto "00:20:30" (1602000, 420000, 22000)
on $NAME moveto "00:21:30" (1590000, 432000, 22000)
xxxEOFxxx

echo " "
echo "CREATING TARGET"
set TYPE=Airplane
set NAME=Base2Plane
set CLASS=Friendly
set LAT=1680000
set LONG=144000
set ALT=0
cronus tropic << xxxEOFxxx
newtype jdltarg
on {jdltarg} create $NAME $TYPE $CLASS "00:14:10" ($LAT, $LONG, $ALT) /noStart
on :jdl:targets CreateEntry \$Object /EntryName=$NAME
xxxEOFxxx
```

```
echo Catalogued target is:  :jdl:targets:$NAME
echo Set AccessControl and Add Segments for that target.

cronus tropic << xxxEOFxxx
on {group} LookupGroup JDLSimulation
on $NAME addtoacl ((\$GroupUID all))
on $NAME moveto "00:14:40" (1674000, 150000, 800)
on $NAME moveto "00:15:30" (1668000, 168000, 1500)
on $NAME moveto "00:16:30" (1662000, 186000, 7000)
on $NAME moveto "00:17:00" (1656000, 198000, 10000)
on $NAME moveto "00:18:00" (1650000, 216000, 15000)
on $NAME moveto "00:18:40" (1644000, 234000, 18000)
on $NAME moveto "00:19:50" (1638000, 252000, 18000)
on $NAME moveto "00:20:20" (1632000, 270000, 18000)
on $NAME moveto "00:21:10" (1626000, 288000, 18000)
on $NAME moveto "00:22:00" (1620000, 306000, 18000)
xxxEOFxxx

echo " "
echo "CREATING TARGET"
set TYPE=Airfield
set NAME=BaseOne
set CLASS=Friendly
set LAT=1707000
set LONG=312000
set ALT=0
cronus tropic << xxxEOFxxx
newtype jdltarg
on {jdltarg} create $NAME $TYPE $CLASS "00:00:30" ($LAT, $LONG, $ALT) /Start
on :jdl:targets CreateEntry \$Object /EntryName=$NAME
xxxEOFxxx

echo Catalogued target is:  :jdl:targets:$NAME
echo Set AccessControl and Add Segments for that target.

cronus tropic << xxxEOFxxx
on {group} LookupGroup JDLSimulation
on $NAME addtoacl ((\$GroupUID all))
on $NAME moveto "00:59:00" (1707000, 312000, 0)
xxxEOFxxx

echo " "
echo "CREATING TARGET"
set TYPE=Airfield
set NAME=BaseTwo
set CLASS=Friendly
set LAT=1680000
set LONG=144000
set ALT=0
cronus tropic << xxxEOFxxx
newtype jdltarg
on {jdltarg} create $NAME $TYPE $CLASS "00:00:30" ($LAT, $LONG, $ALT) /Start
on :jdl:targets CreateEntry \$Object /EntryName=$NAME
xxxEOFxxx

echo Catalogued target is:  :jdl:targets:$NAME
echo Set AccessControl and Add Segments for that target.

cronus tropic << xxxEOFxxx
on {group} LookupGroup JDLSimulation
on $NAME addtoacl ((\$GroupUID all))
on $NAME moveto "00:59:00" (1680000, 144000, 0)
xxxEOFxxx

echo " "
echo "RESTORING ORIGINAL CRONUS WORKINGDIRECTORY"
cronus setwdir $DIR
```

APPENDIX B.4: SAMPLE *initmedsensor* COMMAND DATA FILE

```
# $Revision$
#! /bin/csh
set noglob

echo " "
echo "SAVING CRONUS WORKINGDIRECTORY"
set DIR = `cronus getwdir`

echo " "
echo "CREATE SENSOR OBJECT (MedSea)"
cronus tropic << EOF
on {group} LookupGroupUID JDLSimulation
set cgroup GroupUID
on {jdlsens} AddToAcl ((\$cgroup all))
on {jdlsens} create MedSea radar 1000
set sens obj
on \$sens AddToAcl ((\$cgroup all))
on \$sens MoveTo 0:00:10 0:25:00 0:00:45 ((1260000,120000,0),(1748000,240000,10000)) (1440000,360000,50000)
quit
EOF

echo " "
echo "RESTORING ORIGINAL CRONUS WORKINGDIRECTORY"
cronus setwdir $DIR
exit 0
```

APPENDIX C

APPENDIX C.1: SAMPLE *rootmenu* COMMAND FILE FOR AUTOMATIC STARTUP

```
#
"Command Windows "          MENU
 "Commands "         MENU
  "Command Tool"   cmdtool  -Wf 0 0 0 -WL "" -Wl "Commands"

 "Shell Tool"        shelltool -Wf 0 0 0 -WL ""
 "Commands "         END
 "DOS Window"                 dos    -Wf 0 87 137
 "System Messages"  cmdtool  -Ws 512 128 -Wf 255 200 0 -C -WI /usr/include/images/console.icon -WL ""
"Command Windows "                              END

"JDL"     MENU
 "Set Config"        MENU
  "CECOM-RADC-NOSC"      setconfig config.CERANO
  "CECOM-RADC"  setconfig config.CERA
  "RADC-NOSC"                 setconfig config.RANO
 "Set Config"        END
 "Start Managers"  MENU
  "TriService"       jdlmgrstart all
  "AF Only"                   jdlmgrstart AF
  "ARMY Only"                 jdlmgrstart ARMY
  "NAVY Only"                 jdlmgrstart NAVY
 "Start Managers"  END
 "LinkTo Timer"  MENU
  "ALL"              shelltool -Wf 255 0 0 -Wl "JDL Simulation Time" jdllinktime all
  "Sensor"                    shelltool -Wf 255 0 0 -Wl "JDL Simulation Time" jdllinktime jdlsens
  "TargetSim"                 shelltool -Wf 255 0 0 -Wl "JDL Simulation Time" jdllinktime jdltarg
  "Weather"                   shelltool -Wf 255 0 0 -Wl "JDL Simulation Time" jdllinktime jdlfcast
  "SitReport"                 shelltool -Wf 255 0 0 -Wl "JDL Simulation Time" jdllinktime jdlsit
  "TrackSim"                  shelltool -Wf 255 0 0 -Wl "JDL Simulation Time" jdllinktime jdltrack
 "LinkTo Timer"  END
 "LinkTo Display"   jdllinkdisp
 "Begin Simulation"  MENU
  "TriService"       jdlsimstart all
  "AF Only"                   jdlsimstart AF
  "ARMY Only"                 jdlsimstart ARMY
  "NAVY Only"                 jdlsimstart NAVY
 "Begin Simulation"  END
 "Stop Managers"  MENU
  "TriService"       jdlmgrstop all
  "AF Only"                   jdlmgrstop AF
  "ARMY Only"                 jdlmgrstop ARMY
  "NAVY Only"                 jdlmgrstop NAVY
 "Stop Managers"  END
"JDL"     END
```

```
"Services"                          MENU
 "Redisplay All"                    REFRESH
 "Lock Screen"                      lockscreen
 "Save Screen Layout"  save_screen_layout
 " "   cat /dev/null
 "Display Clock"                    MENU
  </usr/include/images/clock.icon>       clock
  </usr/include/images/clock.rom.icon>   clock -r
 "Display Clock"             END
 "Printing"         MENU
  "Check Printer Queue"  sh -c "echo; echo '-------------'; echo 'Printer queue'; lpq; echo '-------------'@"
  "Print Selected Text"  sh -c "get_selection | lpr ; echo 'Selection printed'.@"
  "Print Screen"                    sh -c "echo 'Saving screen image'.@ ; screendump|rasfilter8to1|lpr -v ; echo 'Screen printed'.@"
 "Printing"         END
 " "   cat /dev/null
 "User Defaults"             defaultsedit -Wf 255 200 0
 "Color Editor"     coloredit -Wf 0 0 0
 "Default Font"  MENU
  "Tiny Font" changefont /usr/lib/fonts/fixedwidthfonts/screen.r.7 "the tiny font"
  "Standard Font" changefont /usr/lib/fonts/fixedwidthfonts/screen.r.13 "the standard font"
  "Bold font" changefont /usr/lib/fonts/fixedwidthfonts/screen.b.14 "the Bold font"
  "Large Font" changefont /usr/lib/fonts/fixedwidthfonts/cour.b.18 "the large font"
 "Default Font"  END
"Services"                          END
"Exits"                             MENU
 "Logout"   EXIT
"Exits"                             END
```

APPENDIX C.2: SAMPLE CONFIGURATION COMMAND FILES FOR AUTOMATIC STARTUP

File Name: config.CERLNO

```
#! /bin/csh -f

# Modified on 07/23/91 by SMH
# Modified on 09/04/91 by L. Dunham for NOSC

setenv TIMER cecom-devi
setenv SIM_DAT nosc-scrappy

setenv WEATHR rl-hydrus
setenv TGTSIM rl-phobos
setenv SENSOR rl-orion
setenv TGTFIL1 rl-orion

setenv TRKSIM nosc-astro
setenv TRKRPT nosc-peabody

setenv SITREP cecom-shiva
```

File Name: config.CENO

```
#! /bin/csh -f

# Modified on 07/18/91 by SMH

setenv TIMER cecom-devi
setenv SIM_DAT nosc-scrappy

setenv WEATHR cecom-vishnu
setenv TGTSIM nosc-peabody
setenv SENSOR cecom-gita
setenv TGTFIL1 cecom-devi
setenv TGTFIL2 nosc-bandit

setenv TRKSIM nosc-peabody
setenv TRKRPT nosc-scooby

setenv SITREP cecom-shiva
```

File Name: config.CERL

```
#! /bin/csh -f

# Modified on 07/18/91 by SMH

setenv TIMER cecom-devi
setenv TRKRPT cecom-gita
setenv SITREP cecom-shiva

setenv SIM_DAT rl-orion
setenv WEATHR rl-hydrus
setenv TGTSIM rl-phobos
setenv SENSOR rl-orion
setenv TGTFIL1 rl-corvus
#setenv TGTFIL2 rl-orion
setenv TRKSIM rl-orion
```

File Name: config.NORL

```
#! /bin/csh -f

# Modified on 07/16/91 by SMH
# Modified on 09/16/91 by LRD

setenv TIMER nosc-bandit
setenv SIM_DAT nosc-peabody

setenv WEATHR rl-hydrus
setenv TGTSIM rl-phobos
setenv SENSOR rl-orion
setenv TGTFIL1 rl-corvus
setenv TGTFIL2 rl-orion

setenv TRKSIM nosc-peabody
setenv TRKRPT nosc-scooby

setenv SITREP nosc-astro
```

File Name: config.CE

```
#! /bin/csh -f

#Modified on 07/18/91 by SMH

setenv TIMER cecom-devi
setenv SIM_DAT cecom-shiva

setenv WEATHR cecom-vishnu
setenv TGTSIM cecom-devi
setenv SENSOR cecom-gita
setenv TGTFIL cecom-devi

setenv TRKSIM cecom-vishnu
setenv TRKRPT cecom-gita

setenv SITREP cecom-shiva
```

File Name: config.RL

```
#! /bin/csh -f

# Modified by SMH 07/16/91

setenv TIMER rl-orion
setenv SIM_DAT rl-corvus

setenv WEATHR rl-hydrus
setenv TGTSIM rl-phobos
setenv SENSOR rl-orion
setenv TGTFIL1 rl-corvus
setenv TGTFIL2 rl-orion

setenv TRKSIM rl-orion
setenv TRKRPT rl-orion

setenv SITREP rl-corvus
```

File Name: config.NO

```
#! /bin/csh -f

# Modified on 07/16/91 by SMH
# Modified on 09/04/91 by L.Dunham

setenv TIMER nosc-bandit
setenv SIM_DAT nosc-scrappy

setenv WEATHR nosc-peabody
setenv TGTSIM nosc-scrappy
setenv SENSOR nosc-scooby
setenv TGTFIL1 nosc-peabody
setenv TGTFIL2 nosc-bandit

setenv TRKSIM nosc-peabody
setenv TRKRPT nosc-scooby

setenv SITREP nosc-astro
```

APPENDIX C.3: SAMPLE *jdlmgrstart* COMMAND FILE FOR AUTOMATIC STARTUP

```
#! /bin/csh -f

set noglob

cd $JDL_ROOT/DEMO/bin
source config

(echo "JDL: Starting Managers") > /dev/console

set startup = $1

# Start the generic services that will always be needed
(echo "  TIMER on $TIMER") > /dev/console
(cronus startservice jdltime @$TIMER) >& /dev/console
(echo "  SIMDATA on $SIMDAT") > /dev/console
(cronus startservice jdldata @$SIMDAT) >& /dev/console

# Start remaining services based on demo conditions
if (($startup == "ALL") || ($startup == "all")) then
 (echo "  WEATHER on $WEATHR") > /dev/console
 (cronus startservice jdlfcast @$WEATHR) >& /dev/console
 (echo "  TARGETSIM on $TGTSIM") > /dev/console
 (cronus startservice jdltarg @$TGTSIM) >& /dev/console
 (echo "  SENSOR on $SENSOR") > /dev/console
 (cronus startservice jdlsens @$SENSOR) >& /dev/console
 (echo "  TRACKSIM on $TRKSIM") > /dev/console
 (cronus startservice jdltrack @$TRKSIM) >& /dev/console
 (echo "  TRACKREPORT on $TRKRPT") > /dev/console
 (cronus startservice jdltrm @$TRKRPT) >& /dev/console
 (echo "  SITUATIONREP on $SITREP") > /dev/console
 (cronus startservice jdlsit @$SITREP) >& /dev/console
else
 switch ($startup)
  case AF :
   (echo "  WEATHER on $WEATHR") > /dev/console
   (cronus startservice jdlfcast @$WEATHR) >& /dev/console
   (echo "  TARGETSIM on $TGTSIM") > /dev/console
   (cronus startservice jdltarg @$TGTSIM) >& /dev/console
   (echo "  SENSOR on $SENSOR") > /dev/console
   (cronus startservice jdlsens @$SENSOR) >& /dev/console
   breaksw
  case ARMY :
   (echo "  SITUATIONREP on $SITREP") > /dev/console)
   (cronus startservice jdlsit @$SITREP) >& /dev/console
   breaksw
  case NAVY :
   (echo "  TRACKSIM on $TRKSIM") > /dev/console
   (cronus startservice jdltrack @$TRKSIM) >& /dev/console
   (echo "  TRACKREPORT on $TRKRPT") > /dev/console
   (cronus startservice jdltrm @$TRKRPT) >& /dev/console
   breaksw
 endsw
endif

(echo "JDL: Managers Started") > /dev/console
```

APPENDIX C.4: SAMPLE *jdllinktime* COMMAND FILE FOR AUTOMATIC STARTUP

```
#! /bin/csh -f

set noglob

cd $JDL_ROOT/DEMO/bin
source config

if (! $?TIMER_UID) then
 echo "JDL:  TIMER_UID variable not set in environment"
 echo -n "Do you wish to establish a simulation timer? (yes,no) "
 set reply = $<
 if ($reply != "yes") exit
else
 echo "Current TIMER_UID is set at: $TIMER_UID"
 echo -n "Do you wish to change the simulation timer? (yes,no) "
 set reply = $<
 if ($reply != "yes") goto SET_MGRS
endif

  echo " "
  echo "Broadcasting for Simulation Timers..."
  cronus tropic "on {jdltime}@broadcast ListObjects"

  echo -n "Enter Timer Selection: "
  set timer = $<
  echo "setenv TIMER_UID $timer" >> config
  setenv TIMER_UID $timer
  SET_MGRS:
  echo " ";echo " "
  echo "Resetting the SimulationTimer"
  (echo "JDL:  Resetting the SimulationTimer") > /dev/console
  tropic "on $TIMER_UID Reset" >& /dev/console

  echo " "
  echo "Linking Managers to SimulationTimer"
  (echo "JDL:  Linking Managers to SimulationTimer") > /dev/console
```

```
set mgr = $1

if (($mgr == "ALL") || ($mgr == "all")) then
 echo "  WEATHER"
 tropic "on {jdlfcast}@$WEATHR LinkToTimer $TIMER_UID"
 echo "  TARGETSIM"
 tropic "on {jdltarg}@$TGTSIM LinkToTimer $TIMER_UID"
 echo "  SENSOR"
 tropic "on {jdlsens}@$SENSOR LinkToTimer $TIMER_UID"
 echo "  TRACKSIM"
 tropic "on {jdltrack}@$TRKSIM LinkToTimer $TIMER_UID"
 echo "  SITREPORT"
 tropic "on {jdlsit}@$SITREP LinkToTimer $TIMER_UID"
else
 switch ($mgr)
  case jdlfcast :
   echo "  WEATHER"
   tropic "on {jdlfcast}@$WEATHR LinkToTimer $TIMER_UID"
   breaksw
  case jdlfcast :
   echo "  TARGETSIM"
   tropic "on {jdltarg}@$TGTSIM LinkToTimer $TIMER_UID"
   breaksw
  case jdlfcast :
   echo "  SENSOR"
   tropic "on {jdlsens}@$SENSOR LinkToTimer $TIMER_UID"
   breaksw
  case jdlfcast :
   echo "  TRACKSIM"
   tropic "on {jdltrack}@$TRKSIM LinkToTimer $TIMER_UID"
   breaksw
  case jdlfcast :
   echo "  SITREPORT"
   tropic "on {jdlsit}@$SITREP LinkToTimer $TIMER_UID"
   breaksw
 endsw
endif

(echo "JDL:  Managers Linked to SimulationTimer") > /dev/console
```

APPENDIX C.5: SAMPLE *jdllinkdisp* COMMAND FILE FOR AUTOMATIC STARTUP

```
#! /bin/csh -f

cd $JDL_ROOT/DEMO/bin
source config

(echo "JDL: Linking Managers to Display") > /dev/console

 (echo "   WEATHER") > /dev/console
  tropic "on {jdlfcast}@$WEATHR LinkToDisplay" >& /dev/console
  (echo "   TARGETSIM") > /dev/console
  tropic "on {jdltarg}@$TGTSIM LinkToDisplay" >& /dev/console
  (echo "   SENSOR") > /dev/console
  tropic "on {jdlsens}@$SENSOR LinkToDisplay" >& /dev/console
  (echo "   TRACKSIM") > /dev/console
  tropic "on {jdltrack}@$TRKSIM LinkToDisplay" >& /dev/console
  (echo "   SITREPORT") > /dev/console
  tropic "on {jdlsit}@$SITREP LinkToDisplay" >& /dev/console
  (echo "   TIMER") > /dev/console
  tropic "on {jdltime}@$TIMER LinkToDisplay" >& /dev/console
  (echo "   SIMDATA") > /dev/console
  tropic "on {jdldata}@$SIMDAT LinkToDisplay" >& /dev/console
  (echo "   TRACKRPT") > /dev/console
  tropic "on {jdltrm}@$TRKRPT LinkToDisplay" >& /dev/console

 (echo "JDL: Managers Linked to Display") > /dev/console
```

APPENDIX C.6: SAMPLE *jdlsimstart* COMMAND FILE FOR AUTOMATIC STARTUP

```
#! /bin/csh -f

set noglob

cd $JDL_ROOT/DEMO/bin
source config

(echo "JDL:  Starting Simulation") > /dev/console

set startup = $1

# Start the simulation clock
(echo "   Starting Simulation Clock") > /dev/console
(tropic "on $TIMER_UID Start") >& /dev/console

# Start services based on demo conditions
if (($startup == "ALL") || ($startup == "all")) then
  (echo "   Starting AF") > /dev/console
  (tropic "on "\":jdl:sensors:${SENSOR}.MedSea\"" operate 0" >& /dev/console)&
  (echo "   Starting NAVY") > /dev/console
  (tropic "on {jdltrm} InitializeDB" >& /dev/console)&
  (tropic "on {jdltrack} ReadTracks navyfile" >& /dev/console)&
  (echo "   Starting ARMY") > /dev/consoie
  (tropic "on {jdlsit} ReadSitReps armyfile" >& /dev/console)&
else
  switch ($startup)
    case AF :
      (echo "   AF Started") > /dev/console
      (tropic "on "\":jdl:sensors:${SENSOR}.MedSea\"" operate 0" >& /dev/console)&
      breaksw
    case ARMY :
      (echo "   ARMY Started") > /dev/console
      (tropic "on {jdlsit} ReadSitReps armyfile" >& /dev/console)&
      breaksw
    case NAVY :
      (echo "   NAVY Started") > /dev/console
      (tropic "on {jdltrm} InitializeDB" >& /dev/console)&
      (tropic "on {jdltrack} ReadTracks navyfile" >& /dev/console)&
      breaksw
  endsw
endif
(echo "JDL:  Managers Started") > /dev/console
```

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE March 1992	3. REPORT TYPE AND DATES COVERED Final
4. TITLE AND SUBTITLE JDL Tri-Service Distributed Technology Experiment		5. FUNDING NUMBERS AN: DN888630 PE: 62232N PROJ: CDB3
6. AUTHOR(S) L. R. Dunham, M. J. Gadbois, and M. F. Barrett		
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Command, Control and Ocean Surveillance Center (NCCOSC) RDT&E Division (NRaD) San Diego, CA 92152–5000		8. PERFORMING ORGANIZATION REPORT NUMBER NRaD TD 2332
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Office of Naval Technology 800 North Quincy Street Arlington, VA 22217–5000		10. SPONSORING/MONITORING AGENCY REPORT NUMBER
11. SUPPLEMENTARY NOTES		
12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.		12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The Tri-Service Distributed Technology Experiment is an unclassified activity initiated under the auspices of the Joint Direc^rs of Laboratories (JDL) by the Networks and Distributed Processing (N&DP) subpanel. The concept of the experiment centers around a demonstrable, short-term, and low-cost computer system application. The intention of the experiment is to overlap technology at each service (Navy, Air Force, and Army) and to use existing resources and internal manpower.

This document describes the functionalities of the tri-service components that make up the experiment and how to demonstrate the experiment. It is assumed that the user is familiar with the Cronus distributed computing environment and its debugging tool, *tropic*.

14. SUBJECT TERMS Tri-Service Distributed Technology Experiment			15. NUMBER OF PAGES 80
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT SAME AS REPORT

UNCLASSIFIED

21a. NAME OF RESPONSIBLE INDIVIDUAL	21b. TELEPHONE *(Include Area Code)*	21c. OFFICE SYMBOL
R. Johnston	(619) 553–4096	Code 413

NSN 7540-01-280-5500

Standard form 298 (BACK)

UNCLASSIFIED

INITIAL DISTRIBUTION

Code 0012	Patent Counsel	(1)
Code 01	R. T. Shearer	(1)
Code 144	V. Ware	(1)
Code 40	R. C. Kolb	(1)
Code 402	J. Maynard	(1)
Code 405	V. J. Monteleon	(1)
Code 41	A. G. Justice	(1)
Code 413	R. E. Johnston	(10)
Code 413	L. Dunham	(30)
Code 413	L. Anderson	(1)
Code 413	M. Butterbrodt	(1)
Code 413	W. Gex	(1)
Code 413	S. Conners	(1)
Code 413	J. Nguyen	(1)
Code 413	C. Pedrano	(1)
Code 413	D. Small	(1)
Code 413	J. Baker	(1)
Code 413	A. Mollet	(1)
Code 413	M. Woodward	(1)
Code 42	R. E. Pierson	(1)
Code 421	M. C. Mudurian	(4)
Code 423	J. P. Schill	(1)
Code 423	R. Crepeau	(1)
Code 4306	R. J. Jaffee	(1)
Code 4601	R. R. Eyres	(1)
Code 752	T. Hufford	(1)
Code 804	R. D. Peterson	(1)
Code 952B	GIDEP Office	(1)
Code 961	Archive/Stock	(6)
Code 964B	Library	(2)

Defense Technical Information Center
Alexandria, VA 22304-6145 (4)

Center for Naval Analyses
Alexandria, VA 22302-0268

Office of Asst Secretary of Defense
Washington, DC 20363-5100 (3)

Office of Naval Technology
Arlington, VA 22217-5000 (2)

DARPA
Arlington, VA 22203-1714 (4)

Space & Naval Warfare Systems Cmd
Washington, DC 20363-5100 (2)

Rome Laboratory/C3AB
Griffis AFB, NY 13441-5700 (3)

NCCOSC Washington Liaison Office
Washington, DC 20363-5100

Navy Acquisition, Research & Development
Information Center (NARDIC)
Washington, DC 20360-5000

Naval Sea Systems Command
Washington, DC 20362-5101

Office of Naval Research
Arlington, VA 22217-5000 (2)

National Security Agency
Fort Meade, MD 20755

Commander in Chief, Pacific Fleet
Pearl Harbor, HI 96860-7000 (2)

U.S. Army HQCECOM
Fort Monmouth, NJ 07703 (2)

INITIAL DISTRIBUTION (Cont'd)

Computer Sciences Corp. Eatontown, NJ 07724	
BBN Systems & Technologies Arlington, VA 22209	(2)
BBN Systems & Technologies Cambridge, MA 02138	(5)
BBN Systems & Technologies San Diego, CA 92110	(2)